The Robots of Dawn

Autores:
90426- Bartolomé, Natalia Giselle
nataliagbar@gmail.com
89575- Durán, Adrián Martín
adrianmdu@gmail.com
92272- Obon, Daniel Andrés
obi.arts@gmail.com

Fecha de entrega: 11/09/2017

Tutor: Prof. Dr. Mariano Mendez

"Facultad de Ingeniería - Universidad de Buenos Aires" 2017

1. Introducción

1.1. Motivación

Los sistemas distribuidos han sido estudiados de forma extensiva sobre ordenadores, sin embargo poco se ha investigado acerca de la implementación de sistemas distribuidos en dispositivos con recursos más limitados que una computadora. Por ello, el desarrollo del presente trabajo se enfoca en presentar un prototipo que sirva para sentar las bases de un futuro desarrollo a gran escala de robots que permitan coordinar su movimiento de forma autónoma. La coordinación de movimiento de dispositivos autónomos es un factor clave cuando tratamos en zonas de difícil o imposible acceso para una persona. Si tomamos como caso de estudio un lugar alejado, regiones de muy difícil acceso para vehículos convencionales, o una selva donde ha ocurrido un accidente aéreo, tener la participación de una red de robots que colaboran en la búsqueda es de gran utilidad. Estos lugares, suelen ser, sitios en los cuales la cantidad de recursos humanos y materiales para desempeñar la tarea de recorrido es muy costosa y compleja de llevar a cabo debido a que la infraestructura tecnológica es escasa también.

Contar con tecnología capaz de conformar una red de trabajo y a partir de la conformación de la misma, permitir a sus nodos interactuar y coordinar trabajo sería de vital importancia.

En la actualidad, el surgimiento del concepto de Internet de las Cosas o "Internet of Things" ha permitido un desarrollo muy acelerado de dispositivos cuyo corazón está conformado por un microcontrolador y posee capacidades de interconexión muy sencillas de usar y aprovechar. Estos dispositivos se denominan placas de desarrollo o "development boards". En los últimos años han surgido, casi explosivamente, un conjunto de development boards que permiten las más variadas características respecto a comunicación, por ejemplo, bluetooth, bluetooth BLE, Wi Fi, conectividad vía LAN o WAN vía RJ45, GSM, GPRS por nombrar algunas de las tecnologías de comunicación a las que se tiene acceso mediante estas placas. También, se estima que para el año 2020, el número de dispositivos conectados crezca en un 50 %, alcanzando los 30 millones, así como también, la información obtenida de sistemas embarcados (los sensores y sistemas que monitorean el mundo físico) crezca hasta un 10 % del universo digital [1].

Habiendo analizado la situación antes mencionada, este trabajo estudiará principalmente:

- La evaluación de la complejidad que presenta la implementación de algoritmos distribuidos en dispositivos IoT, teniendo en cuenta las limitaciones de hardware y software de los mismos.
- La construcción de prototipos de dispositivos autónomos que puedan ejecutar dichos algoritmos.

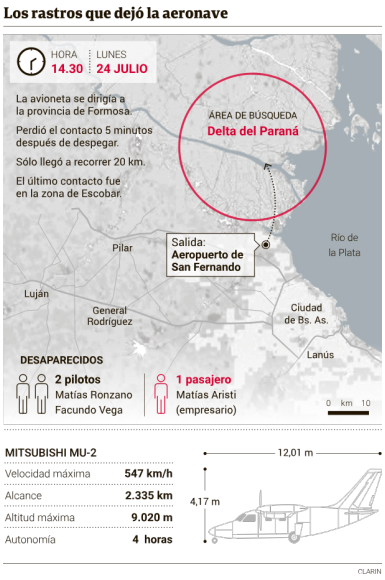
- Realizar una serie de experimentos que demuestran el funcionamiento del algoritmo con el hardware seleccionado.
- Determinar otras posibles áreas de aplicación para la solución propuesta.
- La evaluación de mejoras sobre el/los prototipo/s que perfeccionen la capacidad de coordinación y autonomía de los dispositivos.

Existe una serie de casos de público conocimiento que la tecnología propuesta en este trabajo podría ser (o haber sido) de suma utilidad en el proceso de investigación y esclarecimiento de los mismos. Se presentan a continuación algunos casos donde la utilización de un sistema como el mencionado, hubiera facilitado la tarea de las fuerzas de rescate:

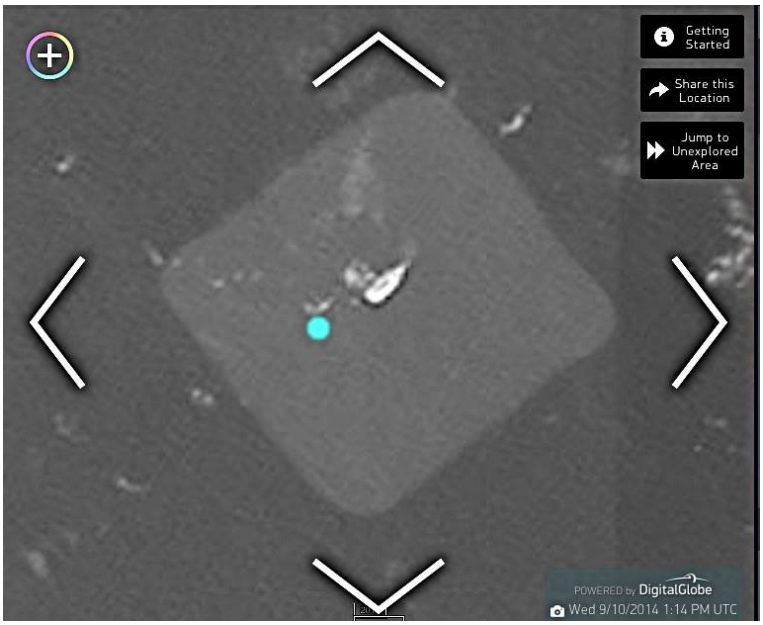
- Caso Pomar[1]: El caso se remonta al año 2009, y trata sobre la desaparición durante 24 días de una familia argentina. Durante la investigación del caso, y ante la falta de pistas concretas de lo acontecido, se manejaron diferentes hipótesis respecto del paradero de la familia[2], siendo un accidente lo que finalmente había acontecido. Sobre el final del caso, diferentes referentes de la investigación aseguraron que la zona en que había ocurrido el accidente, era de difícil acceso y eso complicó la búsqueda de forma significativa.

- Avioneta desaparecida en el Delta[3]: Durante 32 días, se perdió el rastro de una avioneta. La misma había partido el 24 de Julio con tres

pasajeros, y poco tiempo después del despegue se perdió contacto. El ANAC (Asociación Nacional de Aviación Civil) realizo rastrillajes por aire, agua y tierra[4] hasta finalmente hallar restos de la avioneta mediante un avión que sobrevolaba la zona. Las mismas se encontraban semienterradas en un sector pantanoso, en un cráter tapado por agua y vegetación.



- Tunante II[5]: El Tunante II es un velero en el que viajaban cuatro argentinos y que desapareció en aguas brasileñas tras una tormenta. Para la búsqueda, se utilizaron satélites que tomaban fotografías de las zonas en donde podría haber naufragado el velero, para luego analizarlas manualmente.



- Huracán Irma[6]: Entre la noche del 30 de agosto del 2017 y la madrugada del 1 de septiembre, empezó a formarse una de las tormentas más poderosas que ha azotado al Atlántico en un siglo. Presenta una combinación catastrófica de fuertes vientos, oleaje y lluvia que arrasó varias islas del Caribe. Luego de que cedan las tormentas, es importante contar con un mecanismo automatizado que facilite la búsqueda de supervivientes en las zonas castigadas por el huracán.

(Reuters)

(Reuters)

- Terremoto de Chiapas de 2017: Un terremoto de magnitud 8.2 en la escala de Ritcher[7] tuvo epicentro en el golfo de Tehuantepec la noche del jueves 7 de septiembre de 2017. Poco después, ocho países recibían alertas de tsunami[8]. El sismo dejó un saldo de varias decenas

de muertos, así como incontables daños materiales. Al igual que el caso anterior, sería de gran ayuda para las fuerzas de rescate, contar con un mecanismo que permita recorrer las zonas anegadas de la forma más rapida y eficiente posible, sin poner en riesgo la vida de los rescatistas.

1.2. Trabajos Relacionados

1.2.1. Coordinación de IoT

- The coordination of nodes in the Internet of Things [2]: Se propone un método para coordinar nodos IoT. El mismo se divide en dos partes, por un lado lo denominado "Área conocida", donde un coordinador controla el grupo para garantizar el orden en la red, y por el otro, el "Área desconocida" donde se utiliza la nube para buscar y comunicar los nodos.

[1]https://es.wikipedia.org/wiki/Caso_Pomar
[2]http://www.lagaceta.com.ar/nota/354137/Policiales/seis_hipotesis_sobre_familia_Pomar.html
[3]http://www.lavoz.com.ar/sucesos/fin-del-misterio-hallan-restos-de-la-avioneta-desaparecida
[4]https://www.clarin.com/sociedad/avioneta-desaparecida-prefectura-intensifica-busqueda-zona-escobar_0_BkleF90UZ.html
[5]http://www.infobae.com/2015/10/14/1761949-tunante-ii-los-familiares-no-descartan-que-el-velero-haya-sido-secuestrado-piratas
[6]http://www.lanacion.com.ar/2061575-huracan-irma-florida-tormenta-tornado-muerte
[7]https://es.wikipedia.org/wiki/Escala_sismol%C3%B3gica_de_Richter
[8]https://es.wikipedia.org/wiki/Tsunami

- A Revised BROGO Algorithm for Leader Election in Wireless Sensor and IoT Networks [3]: Se toma como base el algoritmo BROGO[9], el cual genera un "Spanning Tree", donde cada hoja del árbol envía un mensaje por su rama hasta la raíz, para determinar, de esta forma quién es el líder de la misma. La raíz del árbol luego elige un líder global a partir de la información recibida de cada una de las ramas. Sin embargo, en este algoritmo puede darse el caso de que el nodo raíz falle antes o durante el proceso de elección de líder. Para evitarlo, se propone que el nodo raíz del Spanning Tree, sea elegido dinámicamente, como aquel nodo que no presente fallas y tenga el mínimo identificador.

- A Networking Perspective on Self-Organizing Intersection Management [4]: Se explora una forma de controlar las intersecciones vehiculares, complementando otras tecnologías de comunicación entre vehículos (Inter Vehicle Communication) como lo son el Wi-Fi o 3G/4G. Las aplicaciones se extienden desde la mejora de la eficiencia del tráfico, hasta situaciones críticas de seguridad, siendo el manejo de intersecciones el aspecto más llamativo. El sistema propuesto "Virtual Traffic Light" (VTL) es una mejora al actual sistema de luces físicas, de forma tal de reducir el tráfico en metrópolis urbanas y el mismo se basa en la coordinación de clusters de autos y la comunicación que mantienen entre ellos.

1.2.2. Redes de nodos para recorrer un área

- Semi-Stochastic Topology Control with Application to Mobile Robot Team of Leader-Following Formation [5]: Un equipo de robots que se muevan siguiendo a un líder, es de gran ayuda cuando se tiene que recorrer de emergencia un area bajo tierra. Este paper propone una topología semi estocástica de control, como una combinación de comunicación y control de movimiento para controlar el equipo de robots. Se demuestra experimentalmente que mediante el control de la topología, se optimiza el rendimiento de las comunicaciones mientras los robots se dirigen a los puntos asignados.

- ResCue[10]: Es un proyecto de drones para asistir en zonas atacadas por catástrofes naturales. El desarrollo usa drones autónomos para explorar en tiempo real, áreas afectadas por catástrofes naturales y ayudando de esta forma a salvar vidas. Los dones exploran en tiempo real el área afectada a través del procesamiento de audio y video, y mediante una plataforma web permite a un equipo de operadores ver los

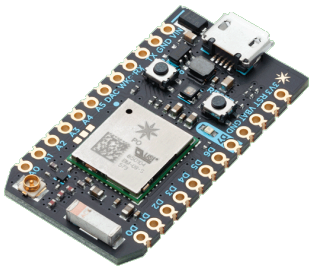
[9] http://pagesperso.univ-brest.fr/ bounceur/anr/persepteur/articles/brogo_dese2017_bounceur.pdf

reportes en simultáneo, ayudando a detectar y reaccionar rápidamente frente a una emergencia[11].

2. Implementación

2.1. Hardware

2.1.1. Particle - Photon



El Photon, creado como una combinación de un poderoso microcontrolador ARM Cortex M3[12] con un chip Wi-Fi de alta velocidad, proporciona todo lo necesario para crear una solución interconectada[13].

Se elige este microcontrolador de código abierto (tanto firmware como hardware) puesto que dado su sistema de tiempo real, se presenta como un dispositivo similar a una computadora a la hora de lidiar con problemas de concurrencia y comunicación, ámbito en el cual poseemos mayor experiencia. Lo antes mencionado, sumado a la versatilidad de un lenguaje de desarrollo como C++, lo posiciona en una situación ideal a la hora de innovar en el ambiente de los sistemas distribuidos con dispositivos IoT.

Especificaciones técnicas:

- Microcontrolador
  - Chip STM32F205 120Mhz ARM Cortex M3
  - 1MB Flash
  - 128KB RAM
  - 18 I/O de uso general

[10] http://www.lanacion.com.ar/2047842-rescue-un-proyecto-de-drones-para-asistir-en-catastrofeS-logro-el-tercer-puesto-en-la-image-cup

[11] https://www.clarin.com/tecnologia/rescue-proyecto-argentino-salvar-vidas-competira-100-mil-dolares_0_BJIueXQb-.html

[12] https://developer.arm.com/products/processors/cortex-m/cortex-m3

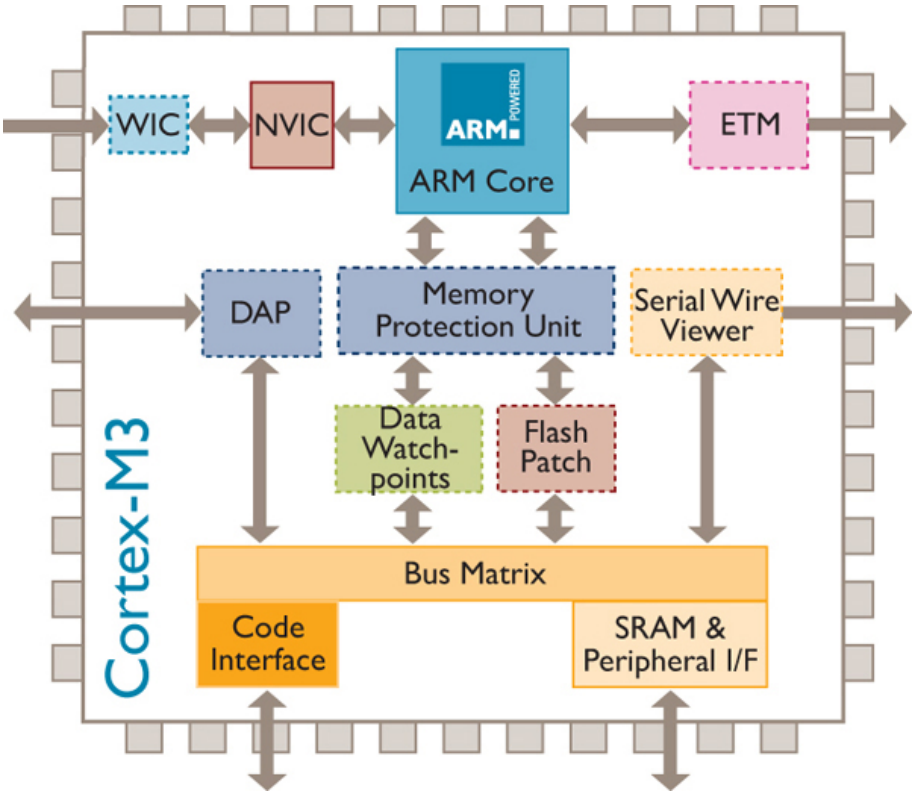
[13] https://www.particle.io/products/hardware/photon-wifi-dev-kit

- Wifi
  - Chip wifi Cypress BCM43362
  - Banda de frecuencia soportada 2.4GHz IEEE 802.11b/g/n
  - Tasas de transferencia de datos inalámbricos de hasta 65Mbit/s
  - Bajo consumo en modo suspendido y detenido
  - Modos de seguridad wifi: Abierta, WEP, WAPI, WPA y WPA2-PSK
  - Configuración como punto de acceso (AP) liviano
- Otras especificaciones
  - Certificación CC/CE/IC/Telec
  - Hardware de código libre

2.1.1.1. Procesador ARM Cortex M3

El procesador ARM Cortex-M3[14] se diseñó para proveer un rendimiento y consumo de energía óptimos. Para lograrlo, el diseño de la arquitectura está basado en el pipeline de 3 etapas Harvard (la cual maximiza el uso de la memoria) y el núcleo ejecuta el set de instrucciones Thumb-2[15].

[14] http://www.embeddedinsights.com/epd/arm/arm-cortex-m3.php
[15] http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0338g/ch01s02s01.html



El procesador integra un núcleo con un controlador de interrupciones dedicado, mejorando significativamente el rendimiento del manejo de interrupciones. El cambio entre interrupciones activas y pendientes, se simplificó mediante el uso de una cola encadenada, en lugar de una pila, pasando de utilizar 30 ciclos de reloj a solo 6. Además, se establecieron tres modos de reposo, lo que mejora significativamente la utilización de energía, incluyendo una función de reposo profundo que puede ser exportada a otros componentes del sistema, permitiendo que el dispositivo entero se pueda apagar rápidamente.

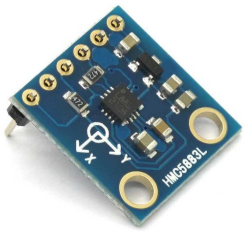
Finalmente, se integran dos componentes opcionales MPU (Memory Protection Unit) y ETM (Embedded Trace Macrocell), los cuales permiten implementar niveles de seguridad en las aplicaciones, separando por proceso, el código, los datos y la memoria.

Especificaciones técnicas:

- Arquitectura Armv7-M Harvard
- Set de instrucciones (ISA) Thumb/Thumb-2
- Pipeline de 3 etapas

- Protección de memoria opcional mediante una unidad de protección de memoria de 8 regiones, subregiones y una región de segundo plano
- Interrupciones: NMI y de 1 a 240 interrupciones físicas
- Niveles de prioridad de las interrupciones: 8 a 256 niveles de prioridad
- Controlador de interrupción de activación: Hasta 240 interrupciones de activación
- Modos de reposo: Instrucciones WFI y WFE integradas, señales de reposos y reposo profundo. Modo de retención opcional con el kit de manejo de energía de ARM
- Manipulación de bits: Instrucciones integradas y banda de bits.
- Instrucciones mejoradas: División por hardware (1-12 ciclos), multiplicación en un ciclo.
- Debug: Hasta 8 breakpoints y 4 watchpoints.
- Rastreo: Rastreo de instrucciones y datos opcionales mediante ETM[16]

2.1.2. Brújula digital HMC5883L



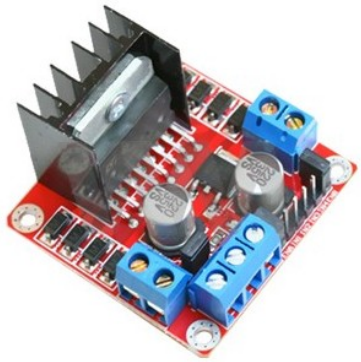
El módulo HMC5883L[17] es un magnetómetro de 3 ejes, diseñado para obtener tanto la dirección como la magnitud de las componentes del campo magnético de la Tierra. De esta forma, conociendo la dirección del campo magnético terrestre, podemos calcular la orientación del dispositivo con respecto al norte magnético de la Tierra. Entre las aplicaciones más frecuentes del módulo se incluyen teléfonos móviles, notebooks, sistemas de navegación automática y dispositivos de navegación personal.

Especificaciones técnicas:

[16] https://developer.arm.com/products/system-ip/coresight-debug-and-trace/coresight-components/system-trace-macrocell
[17] https://cdn-shop.adafruit.com/datasheets/HMC5883L_3-Axis_Digital_Compass_IC.pdf

- Sensor magneto-resistivos de 3 ejes de alta resolución
- Conversor analógico digital de 12 bits
- Interface digital I2C
- Operaciones de baja tensión (2,16 a 3,6V)
- Bajo consumo de corriente (100 μA)
- Superficie de 3,0 x 3,0 x 0,9 mm

2.1.3. Puente H



El puente H, conformado por el chip L298N[18] , permite controlar dos motores de corriente continua o un motor paso a paso bipolar de hasta 2 amper. El módulo cuenta con todos los componentes necesarios para funcionar sin necesidad de elementos adicionales, entre ellos diodos de protección y un regulador LM7805 que suministra 5V a la parte lógica del integrado L298N. Cuenta con jumpers de selección para habilitar cada una de las salidas del módulo.

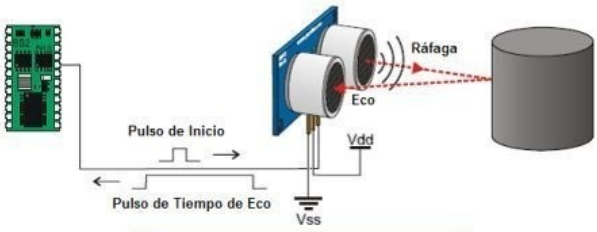
Especificaciones técnicas:

- Tensión de entrada: +5V a +46V
- Tensión de control / Tensión de salida:
  - Nivel bajo: -0.3V a 1.5V
  - Nivel alto: 2.3V a Vss
- Potencia máxima: 25W (a 75°C)
- Temperatura de trabajo: -25°C a +130°C
- Tamaño: 60 x 54 mm

[18] http://www.alldatasheet.com/datasheet-pdf/pdf/22440/STMICROELECTRONICS/L298N.html

2.1.4. Sensor de Proximidad HC-SR04

El módulo de alcance ultrasónico HC-SR04[19] permite detectar objetos en el rango de 2cm a 400cm con una precisión que puede llegar a 3 mm. El módulo incluye transmisores ultrasónicos, receptor y circuito de control.



El principio básico es el siguiente:

- Un pulso ultrasónico corto se transmite en el instante 0.
- Al chocar con un objeto, el pulso es reflejado y recibido nuevamente por el sensor, el cuál lo convierte en una señal eléctrica.
- El siguiente pulso puede ser transmitido cuando el eco del mismo se desvanece.
- Midiéndose el la duracion del proceso recien mencionado puede saberse la distancia entre el sensor y el objeto detectado

Especificaciones técnicas:

- Tensión de trabajo: +5V
- Corriente de trabajo: 15mA

[19] https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf

- Frecuencia: 40Hz
- Rango de medición: 2 cm a 4 m
- Ángulo de medición: 15°
- Señal del disparador: 10 µs (pulso TTL)
- Dimensiones: 45 x 20 x 15 mm

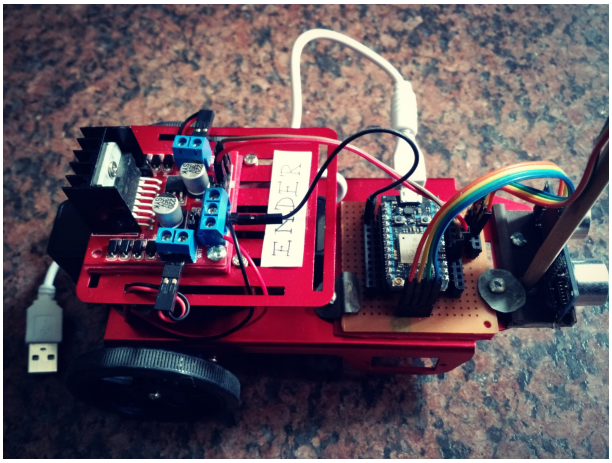
2.1.5. Mini Rover Robot Chassis Kit

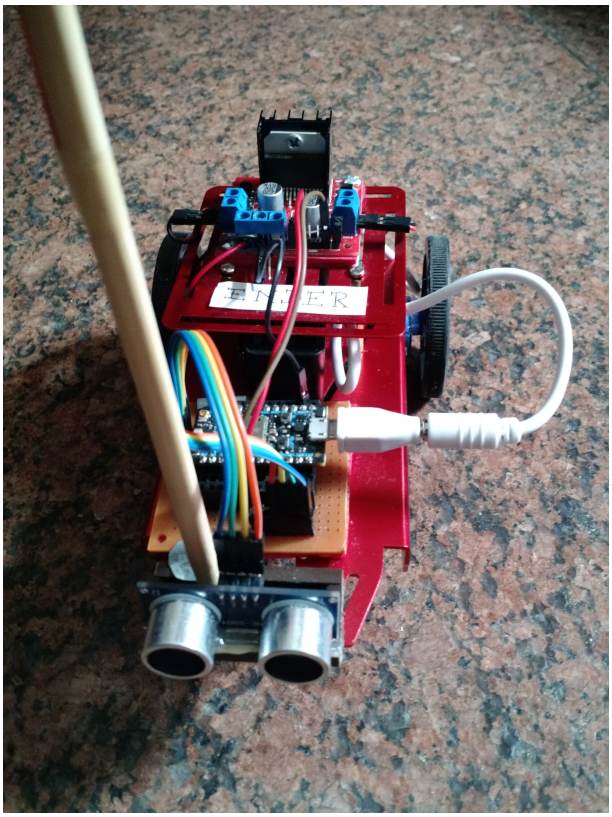
El kit posee todo lo necesario para construir un Robot Rover de 3 ruedas. El mismo se compone de un chasis de aluminio, ligero y fuerte, 2 ruedas de silicona alimentadas por motores de corriente continua, y una rueda de soporte que permite una rotación de 360°.

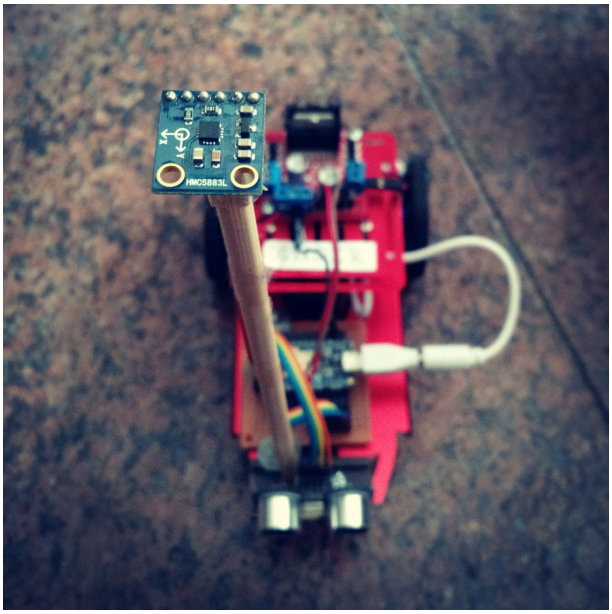
(Adafruit)

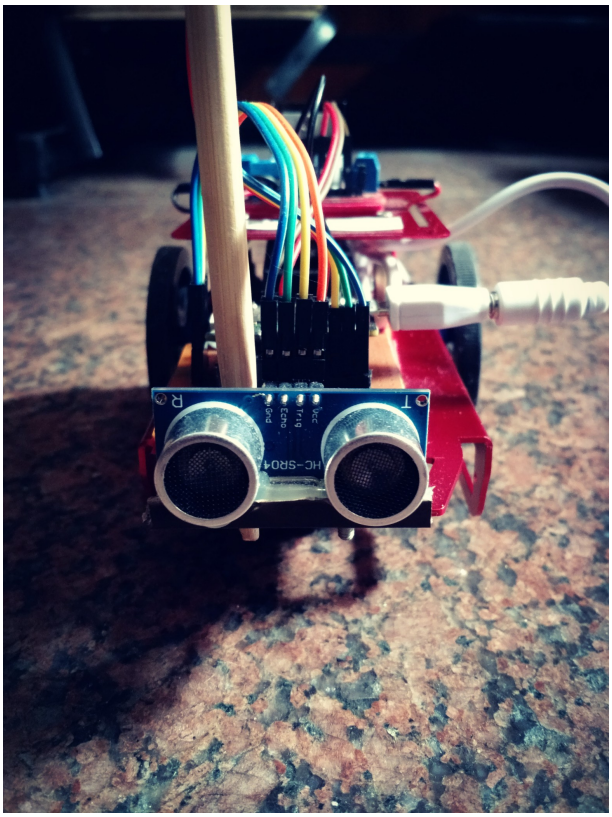
(Adafruit)

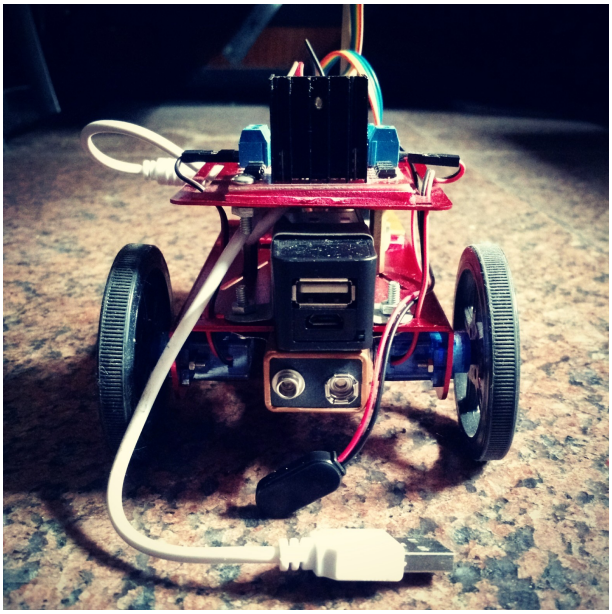
A continuación, se observa imágenes del robot ensamblado conteniendo microcontrolador, puente H, sensor ultrasónico y brújula. La brújula se encuentra alejada del robot mismo para evitar la interferencia generada por el accionar de los motores, así como también, por el chasis de aluminio del robot. La alimentación está separada, por un lado se alimenta el puente H y los motores mediante una batería de 9V, mientras que el microcontrolador, el sensor ultrasónico y la brújula son alimentados por un banco de carga USB.

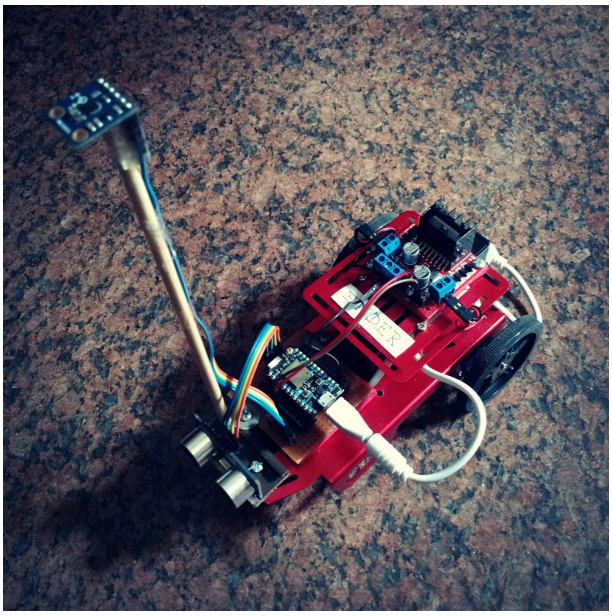





HMC5883L


R
T
Gnd
Echo
Trig
Vcc
HC-SR04



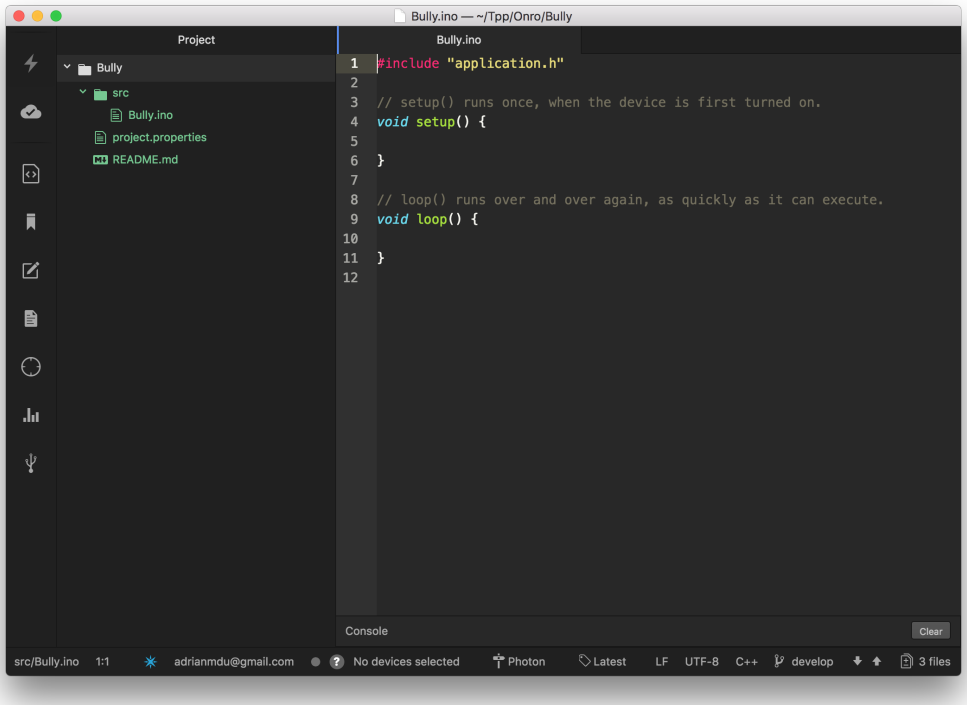


2.2. Software

2.2.1. Atom - Particle Web IDE

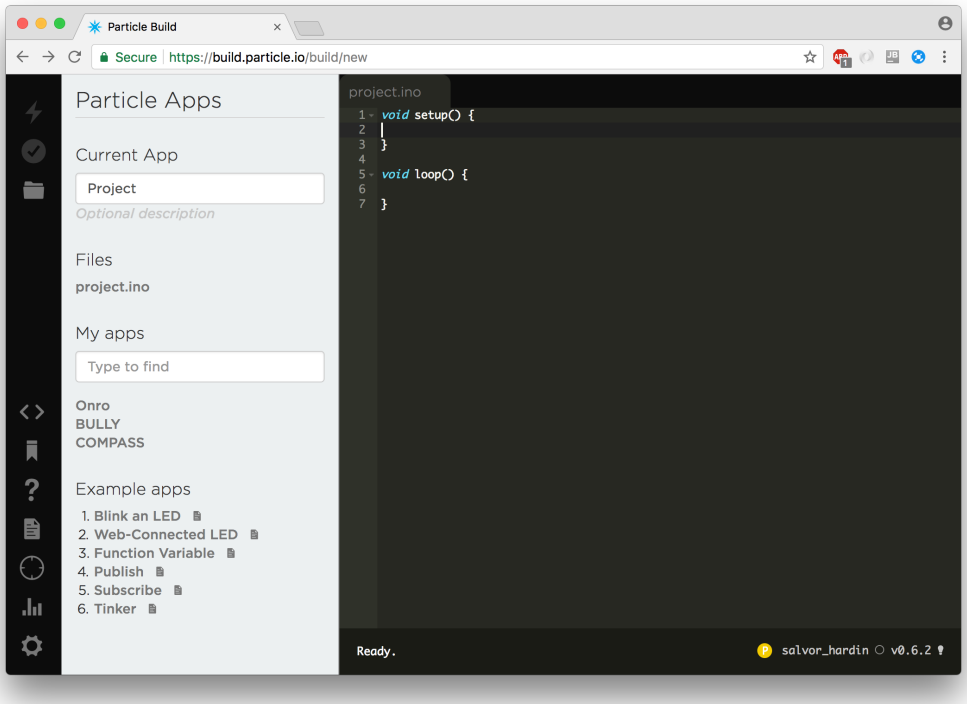
Para el desarrollo del sistema, la empresa fabricante de los dispositivos "Particle"[20], ofrece dos entornos. Por un lado, se puede utilizar "Atom" [21], un editor de texto personalizable a través de una serie de plugins que se le pueden agregar.

[20] http://particle.io
[21] https://atom.io/



O también se puede utilizar un entorno web [22] provisto por Particle.

[22] https://build.particle.io/build



Entre las herramientas provistas por el entorno podemos destacar:

- Compilado del código fuente en la web.
- Visualización de los dispositivos IoT asociados a la cuenta registrada, distinguiendo aquellos que se encuentran en línea.
- Carga del firmware generado en el dispositivo elegido.
- Monitor serie.
- Manejo de repositorio de código.

2.2.2. Lenguaje de desarrollo

Como se mencionó anteriormente, el lenguaje de desarrollo es un subset de C++, complementado por una colección de bibliotecas de código abierto disponibles.

Además del software desarrollado para los microcontroladores, se utilizó el lenguaje Java[23] para realizar prototipos de los algoritmos diseñados. Se eligió Java como lenguaje de alto nivel dado el conocimiento previo que los integrantes del equipo tienen sobre el mismo, así como también, la rapidez que brinda dicho lenguaje a la hora de hacer pruebas de concepto.

[23] https://www.java.com

2.2.3. Algoritmos Implementados

2.2.3.1. Algoritmo de elección de líder

Se diseñó un algoritmo de elección de líder basado en el "Bully" [6], en el cual, ante la falta o caída del líder, todos los integrantes de la red deben elegir un nuevo coordinador. Al iniciar el algoritmo, el dispositivo se asume líder (mismo principio utilizado por el protocolo Spanning Tree[24] para elegir un puente raíz) y envía un mensaje de multidifusión con su prioridad al resto. Al recibir un mensaje, los integrantes de la red lo comparan con la prioridad del dispositivo que consideran líder. Si la prioridad recibida es mejor que la almacenada, se considera el emisor de dicho mensaje como líder, caso contrario, si el receptor del mensaje es el líder, se envía un mensaje con su propia prioridad para notificar del error a los dispositivos que enviaron el mensaje.

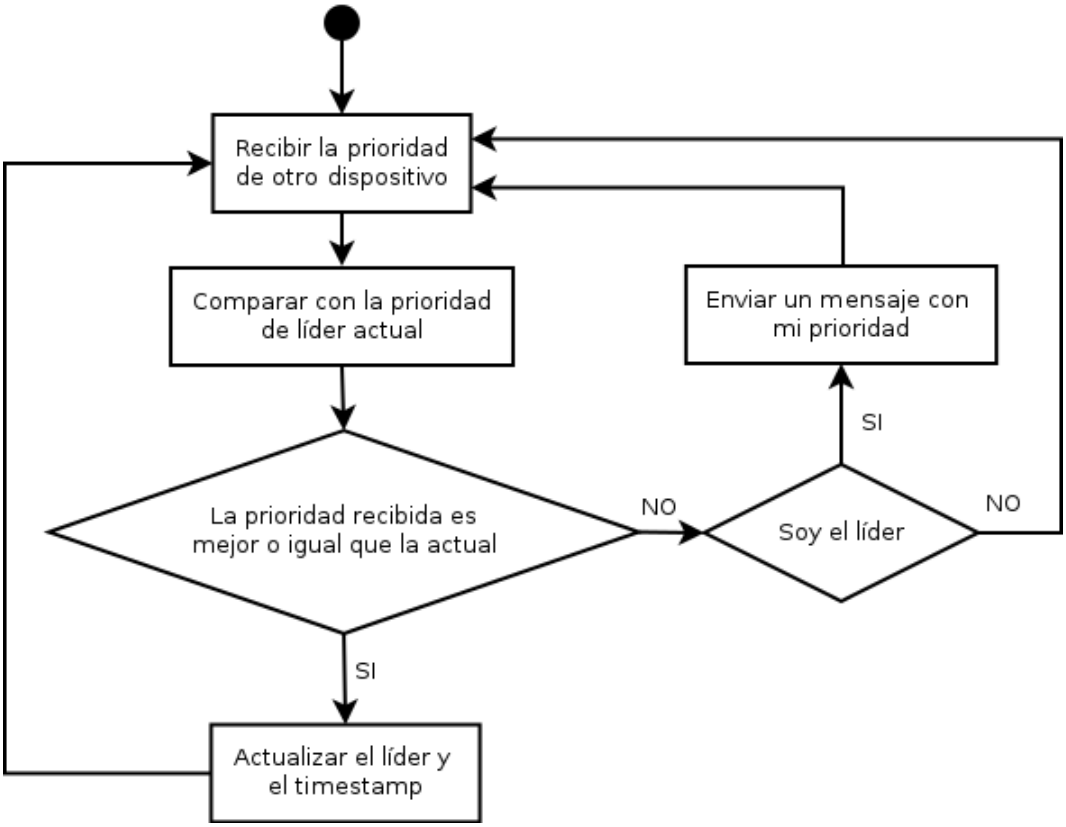
Se utilizan mensajes de multidifusión (o multicast[25] para permitir que cualquier nuevo dispositivo pueda sumarse a la red sin restricciones del momento en que lo hace. El mensaje consta de la dirección física de los dispositivos (MAC) utilizada como prioridad. La elección del líder finalmente se basa en la comparación y selección del integrante con mejor prioridad.

Para la implementación del algoritmo se definieron tres procesos cada uno con una tarea específica:

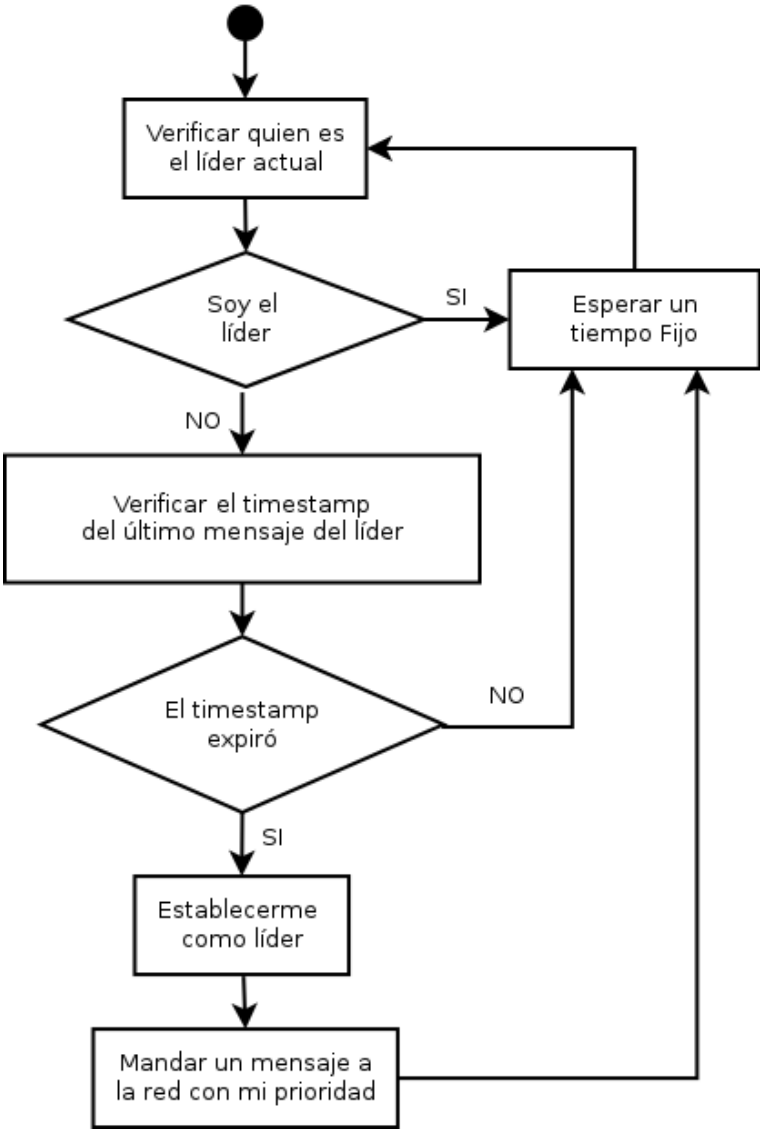
Bully Receiver : Proceso encargado de recibir constantemente los mensajes pertinentes a la elección del líder. Al recibir un mensaje del tipo Bully, el proceso analiza la prioridad y el origen del mensaje. Si la prioridad recibida es mejor que la almacenada, entonces se actualiza el líder actual por el recibido en el mensaje. Si la prioridad es menor, y el proceso se considera líder, entonces le responde a toda la red con su propia prioridad. Caso contrario, si la prioridad recibida es menor que la almacenada, y además, el proceso no se considera líder, simplemente descarta el mensaje, a la espera de que el líder correspondiente responda.

[24] https://www.cisco.com/c/es_mx/support/docs/lan-switching/spanning-tree-protocol/5234-5.pdf

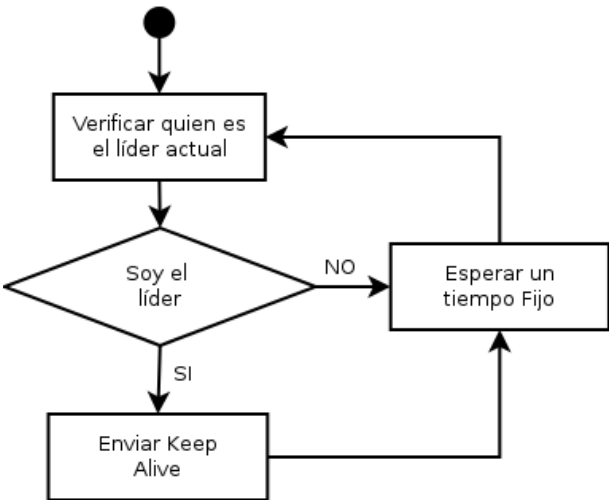
[25] https://www.iana.org/assignments/multicast-addresses/multicast-addresses.xhtml



Liveness Verifier : El algoritmo funciona esencialmente por medio de temporizadores. Al elegir un nuevo líder o recibir un mensaje del mismo, cada proceso inicia un temporizador. Si el temporizador expira sin que se haya recibido un mensaje por parte del líder, entonces se asume que el líder está caído, y se debe elegir uno nuevo, dando comienzo al algoritmo nuevamente. Este proceso es el encargado de verificar el temporizador e iniciar el algoritmo nuevamente de ser necesario.



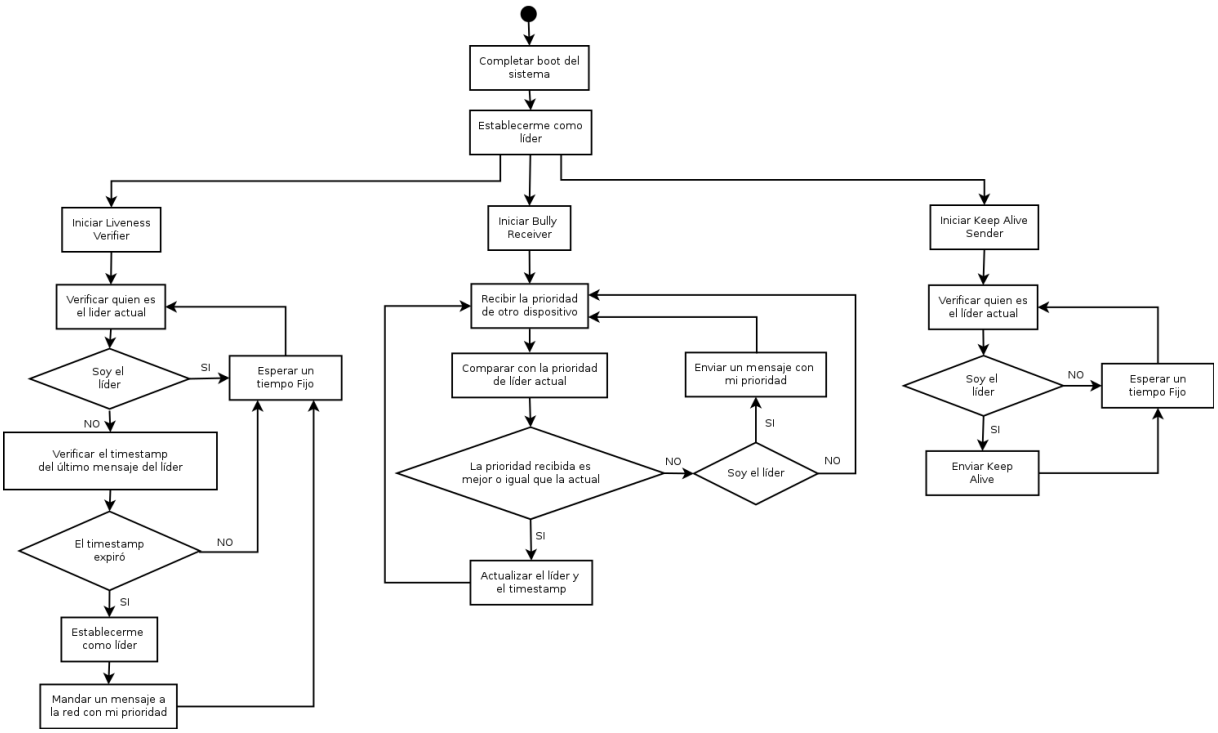
Keep Alive Sender : Una vez elegido un líder, la red de dispositivos alcanza un estado estable en donde no se siguen enviando mensajes. Por lo mencionado en el párrafo anterior, el líder, debe seguir enviando mensajes de "Sigo Vivo" (Keep Alive) de forma regular, para que el resto de la red no consideré que el mismo está caído.



Como puede verificarse en los diagramas, todos los procesos definen un estado inicial, pero no uno final. Esto se debe a que el algoritmo se pone en marcha cuando el dispositivo se enciende, y continúa iterando mientras el dispositivo esté activo.

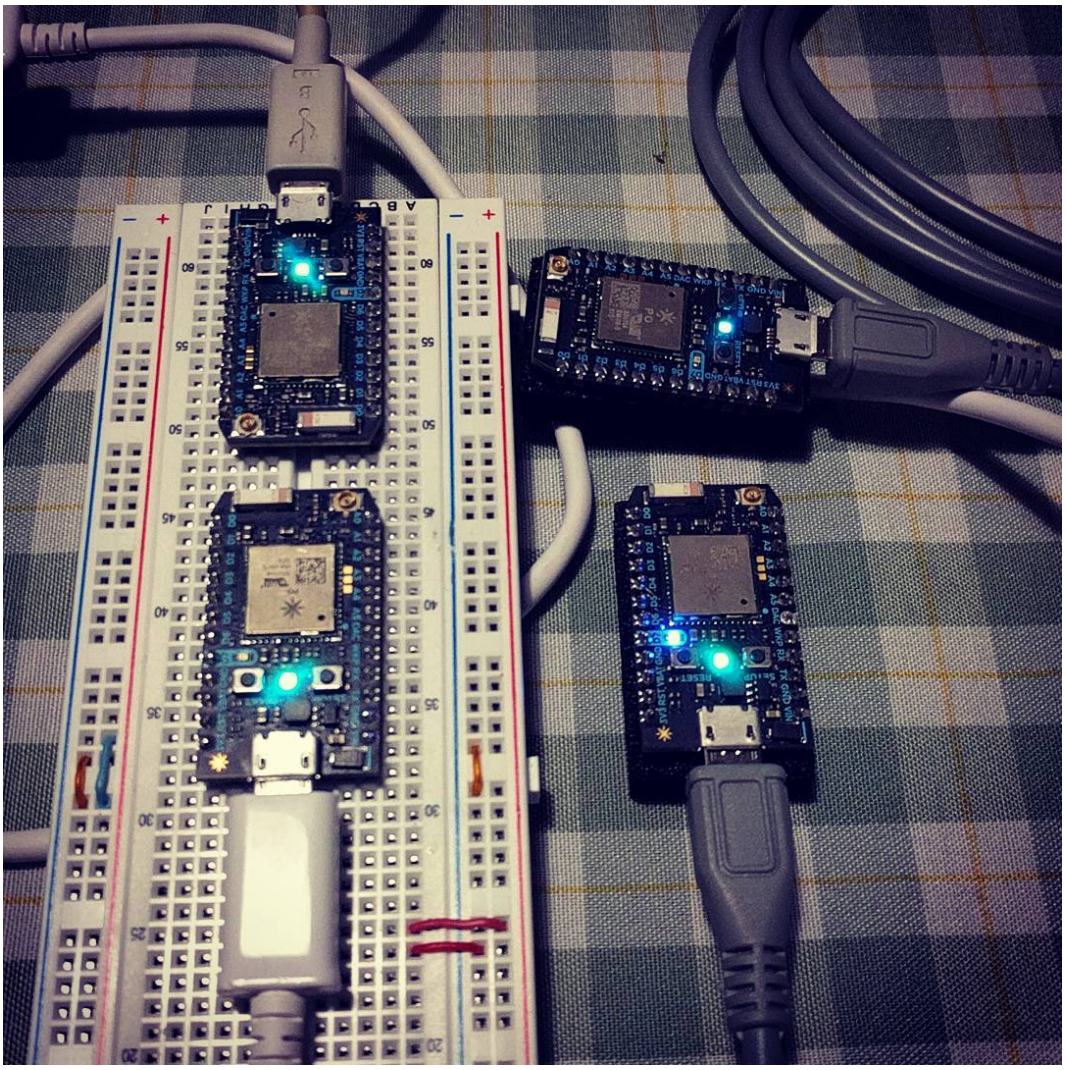
Antes de comenzar con el algoritmo de coordinación de movimiento, los dispositivos esperan un tiempo de estabilización, permitiendo que todos los integrantes sepan quién es el líder. Esto se realiza, puesto que dada la naturaleza del algoritmo, al principio todos los dispositivos se consideran líderes, y de no esperar este tiempo, todos van a coordinar al resto de la red de forma inconsistente.

A continuación se muestra un diagrama completo de la ejecución del algoritmo, como la integración de los tres procesos antes mencionados.



Para la visualización del líder, se hace uso de un led, el cual se enciende cuando un dispositivo se considera líder. A continuación se muestra una red de microcontroladores, donde puede verse que tres de los dispositivos tienen un único led encendido (el cual indica mediante un código de color que el microcontrolador está funcionando correctamente[26], mientras que el cuarto tiene encendido el led mencionado y el que indica que dicho microcontrolador fue elegido como líder.

[26]https://docs.particle.io/support/troubleshooting/troubleshooting-support/photon



2.2.3.2. Algoritmo de coordinación de movimiento

El algoritmo de coordinación de movimiento es ejecutado sólo por el líder, una vez que se estabilizó la elección y la red se coordinó en la elección de un único líder.

El objetivo del mismo es asignar las direcciones a la cual los integrantes de la red deben dirigirse, para de esta manera, cubrir la mayor superficie posible. Para lograrlo, el líder cuenta con una lista con los dispositivos de la red y las direcciones a las que se están dirigiendo. Dichas direcciones son obtenidas por medio de la brújula que posee cada dispositivo y enviadas mediante un mensaje de multidifusión junto con la dirección IP[27] del dispositivo. Se utilizan mensajes de multidifusión para que todos los dispositivos

[27] https://tools.ietf.org/html/rfc791

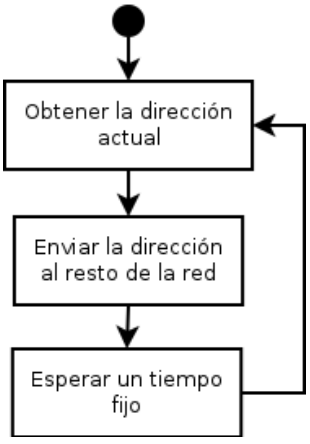
de la red y no solo el líder tengan la información antes mencionada.

Frente a la caída del líder, luego de que la red elija un nuevo coordinador, este último ya cuenta con toda la información necesaria para que el algoritmo de coordinación de movimiento pueda funcionar correctamente, sin la necesidad de esperar a que todos los demás dispositivos le envíen su dirección.

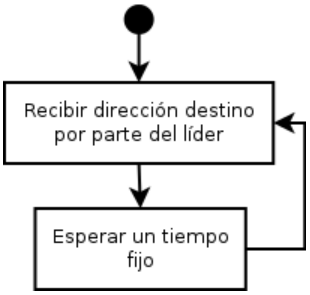
A partir de la información almacenada, el líder asigna a cada integrante la dirección en la que debe moverse. Esto se realiza dividiendo 360°por la cantidad de dispositivos disponibles en la red, y asignando a cada uno, la dirección más cercana a la que tiene originalmente. Finalmente, el líder envía un mensaje de manera directa (Unicast) a cada dispositivo, con la dirección a la que debe moverse.

Periódicamente, se ejecuta el algoritmo de coordinación, reasignándose las direcciones, de forma tal de que en caso de que un dispositivo haya cambiado de dirección y se encuentre más próximo que otro, se le asigne la dirección más cercana. Para el funcionamiento del algoritmo de coordinación de movimiento intervienen tres procesos (en el prototipo original se definieron cuatro procesos, pero dada la naturaleza de los microcontroladores, se decidió juntar dos de los procesos, para de esta forma, consumir menor cantidad de recursos y reducir el acceso a los recursos compartidos):

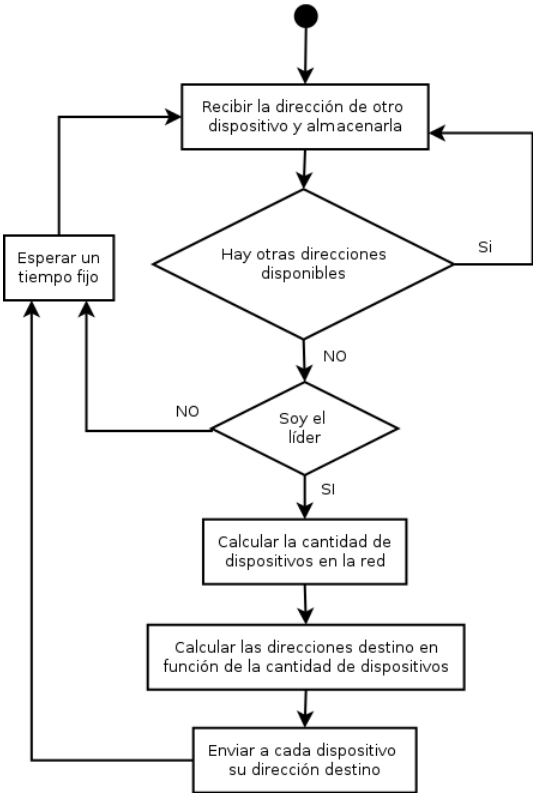
Compass Direction Sender : Es el proceso encargado de enviar por medio de un mensaje de multidifusión, la dirección actual del robot.



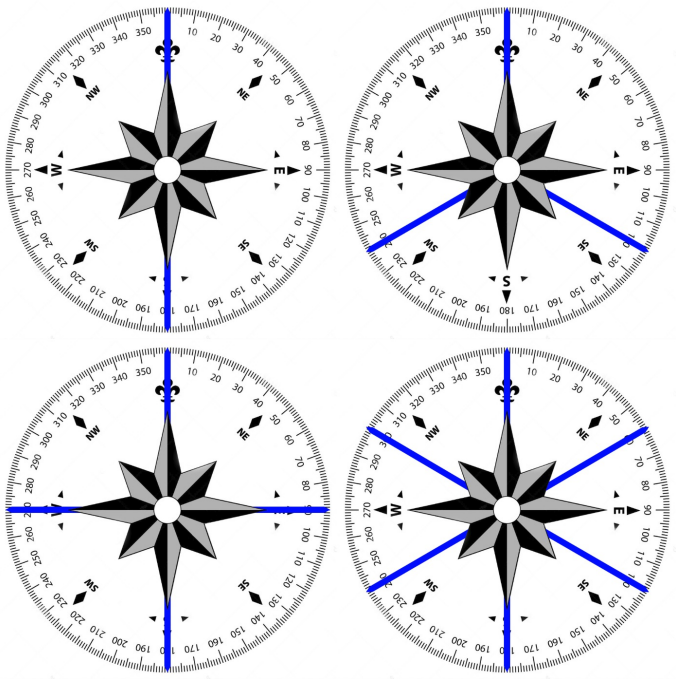
Movement Direction Receiver : Una vez que el líder definió las direcciones que debe tomar cada uno de los robots, les envía mensajes individuales, los cuales son recibidos en este proceso.



Movement Coordination : Originalmente se definió un proceso encargado de recibir las direcciones que envían los dispositivos y otro encargado de la coordinación misma. Sin embargo, en versiones posteriores del desarrollo, se los junto en este único proceso. El mismo recibe todas las direcciones a la que se dirigen los dispositivos de la red y en base a las mismas y la cantidad de dispositivos disponibles, calcula e informa las direcciones que deben tomar los robots para cubrir la mayor superficie posible. Para el cálculo de la dirección destino simplemente se calculan los ángulos que deben tomar los dispositivos como 360°/cantidad de dispositivos, y se asigna cada dirección a aquel dispositivo que esté más cerca de dicha dirección.



Si la red cuenta con tres dispositivos A con dirección 178°, B con dirección 39°y C con dirección 270°, el líder definirá como destino las direcciones 0°, 120°y 240°, enviando al dispositivo B a la dirección 0°, al dispositivo A a la dirección 120°, y finalmente el dispositivo C a la dirección 240°. A continuación, se muestran las direcciones que tomaran los robots, según si la red está compuesta por dos, tres, cuatro o seis dispositivos respectivamente.



2.2.3.3. Algoritmo de movimiento

Se aplica un algoritmo de posicionamiento para orientar al dispositivo a partir de la dirección leída en la brújula (dirección a la que está orientado) y la provista por el líder (dirección a la que el dispositivo debe dirigirse).

Este algoritmo compara ambas direcciones y decide si debe girar para posicionarse tomando el camino más corto posible, moviéndose sobre su propio eje, hasta que la dirección medida por la brújula sea igual a la indicada por el líder con una cierta tolerancia.

Una vez posicionado, el dispositivo avanza midiendo periódicamente la dirección a la que se está dirigiendo y aplica correcciones a su dirección si detecta que se desvió de la dirección deseada. En este caso, la tolerancia es mayor a la original, para evitar de esta forma un efecto rebote, en el cual el dispositivo está constantemente corrigiendo la dirección, y nunca avanza.

El algoritmo de movimiento también posee un sensor de proximidad, el cual actúa en paralelo durante el avance del dispositivo y detiene los

motores en caso que se detecte un obstáculo a una distancia considerada como peligrosa.

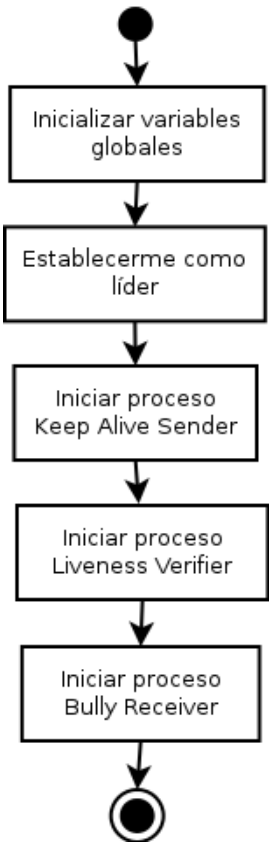
La ejecución del algoritmo de movimiento se realiza íntegramente en el proceso principal del microcontrolador, puesto que el mismo lee constantemente los valores provistos por la brújula y el sensor de proximidad ultrasónico, y es recomendable que dichos valores se obtengan en la ejecución del proceso principal.

2.2.3.4. Ciclo de vida del dispositivo

El firmware del dispositivo nos provee de dos funciones con fines específicos. La función de "setup" es un componente estándar de cualquier programa de microcontroladores, se ejecuta una única vez cuando dispositivo arranca o se resetea. Luego, tenemos la función de "loop" (ciclo), la cual también es esencial en cualquier programa de microcontrolador, la misma se repite una y otra vez tan rápido y tantas veces como le sea posible, luego de que la función "setup" sea invocada. Es importante destacar que si la ejecución dentro del "loop" toma demasiado tiempo, el funcionamiento del dispositivo no está definido, y pueden ocurrir eventos arbitrarios no esperados, como por ejemplo la pérdida de conexión de red. Para evitar este comportamiento errático, se pueden utilizar demoras programadas de un determinado tiempo, durante la cual se frena la ejecución del proceso en cuestión.

2.2.3.4.1. Setup

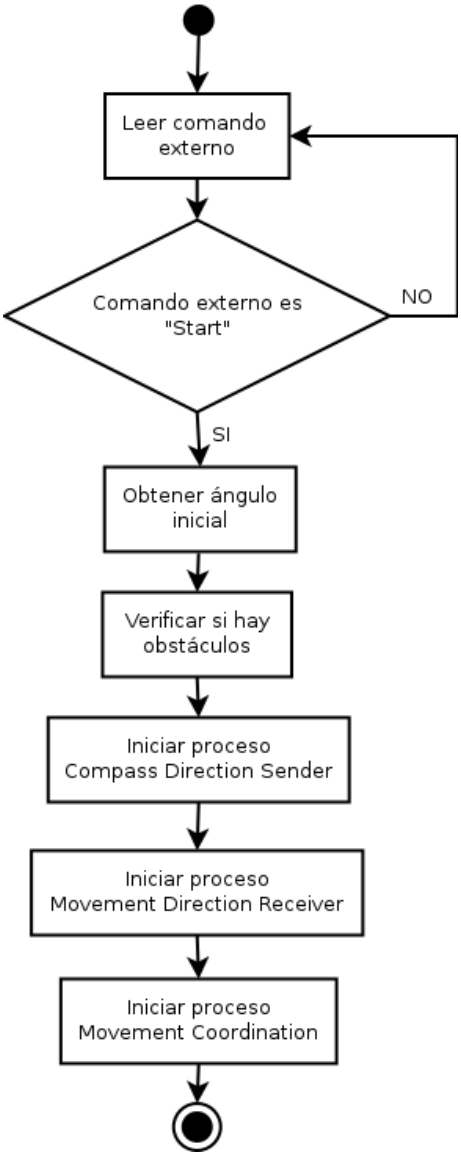
Durante el setup del sistema, se procede a configurar los valores iniciales de las variables involucradas en el funcionamiento del mismo, así como también, se da inicio a los procesos involucrados en el algoritmo de detección de líder. Además de lo mencionado, se realizan tareas de inicialización de los pines de entrada y salida utilizados por el sistema y la creación del sistema de registro. A continuación se muestra un gráfico con el flujo de inicio.



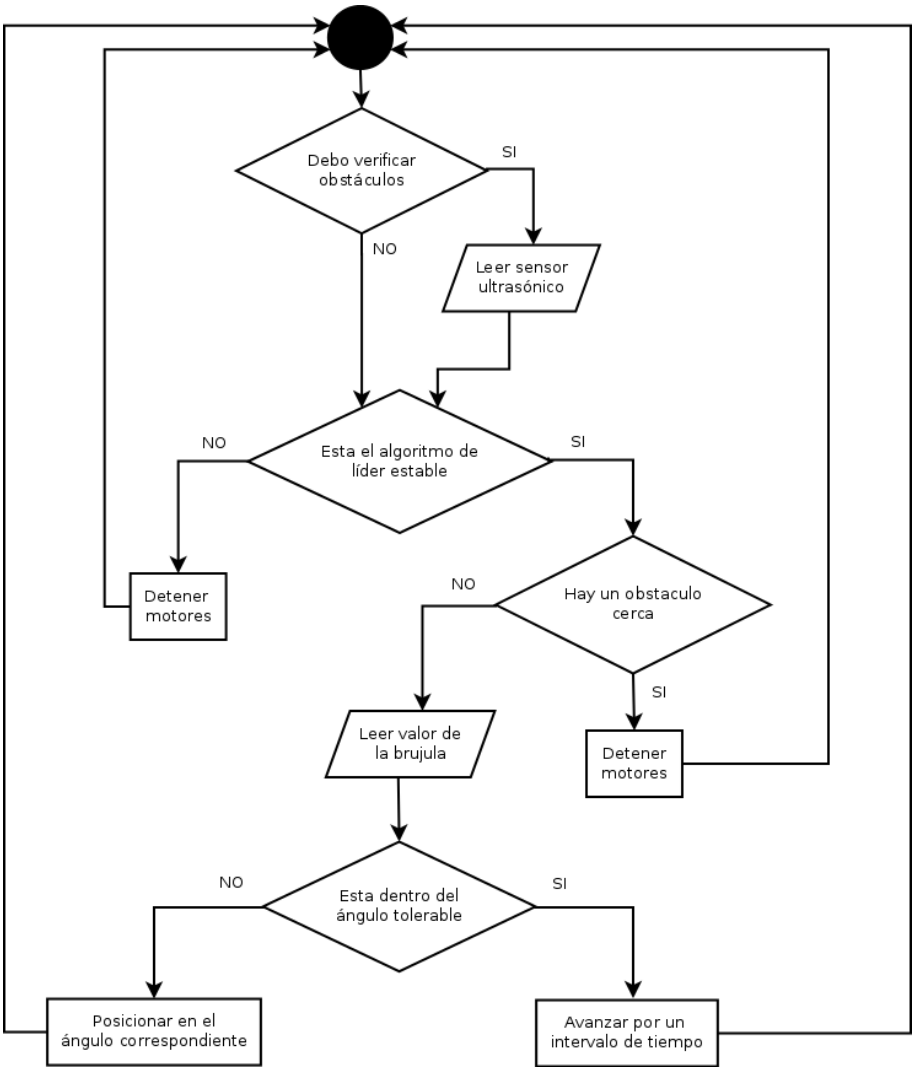
2.2.3.4.2. Loop

El proceso de Loop es más complejo que el setup, puesto que es el encargado de gobernar el comportamiento del sistema. Con fines académicos, el proceso de loop se lo separó en dos partes, una de inicialización y la otra de ejecución.

La parte de inicialización se la colocó en el proceso de loop y no en el setup, puesto que de esta forma, es más fácil de visualizar la ejecución de los diferentes algoritmos del sistema. Este bloque de código queda a la espera de un comando externo para inicializar la ejecución. Una vez recibido el comando, se lanzan los procesos que coordinan el comportamiento del movimiento de toda la red y se procede a la siguiente etapa sin volver a ejecutar el código de inicialización. Al igual que el proceso de setup, la etapa de inicialización tienen un inicio y fin definidos.



En cada iteración del Loop, se leen los valores obtenidos de la brújula del sistema y el sensor ultrasónico, para poder de esta manera, posicionar el robot en la dirección que le corresponde, y avanzar hacia ella, siempre y cuando no se encuentre un obstáculo en el camino.



3. Pruebas

Dada la naturaleza concurrente que presenta un sistema distribuido, es imposible predecir o siquiera obtener el estado global del sistema en un determinado momento. Esto limita significativamente las herramientas de pruebas presentes actualmente en el mercado, siendo posibles únicamente las pruebas exploratorias del sistema, donde queda a cargo de un desarrollador la ejecución y posterior análisis de los procesos que integran el sistema.

Sumado a lo mencionado en el párrafo anterior, se trabajó en el desarrollo de un sistema, el cual es ejecutado sobre microcontroladores con un hardware específico, y limitado en sus formas de comunicación de los resultados.

Por estos dos motivos, se decidió armar prototipos del sistema en un lenguaje de alto nivel, que permitiese de forma más intuitiva analizar el

funcionamiento de los algoritmos, abstrayendo la complejidad del microcontrolador.

Una vez implementados y probados los prototipos, se pasó a la siguiente etapa, implementando los algoritmos sobre los dispositivos finales, y se verificó el correcto funcionamiento armando una red entre el prototipo desarrollado en lenguaje de alto nivel y el microcontrolador.

Finalmente, se removió el prototipo de la red, y se dejó los microcontroladores funcionando autónomamente. Para verificar el correcto funcionamiento, se implementó un sistema de registro remoto, el cual permitía analizar en un pseudo tiempo real el estado de cada uno de los dispositivos.

3.1. Microcontroladores

Las pruebas de los algoritmos integrados en los microcontroladores se realizaron íntegramente mediante el análisis de logs obtenidos de forma remota. Para la exploración, se definió una serie de casos de pruebas los cuales debían ser exitosos luego de un determinado tiempo de estabilización del algoritmo. A continuación se detallan los casos de prueba según el algoritmo que se está probando:

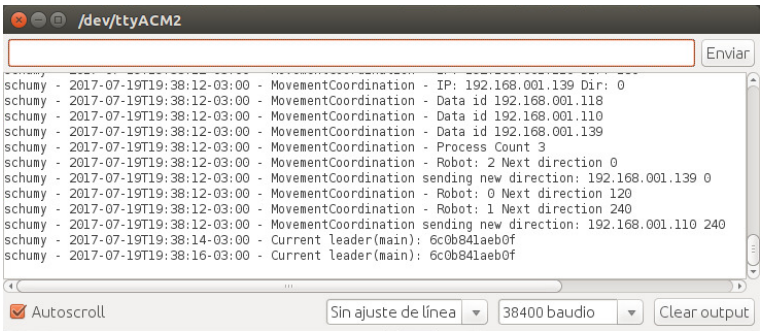
3.1.1. Detección de líder

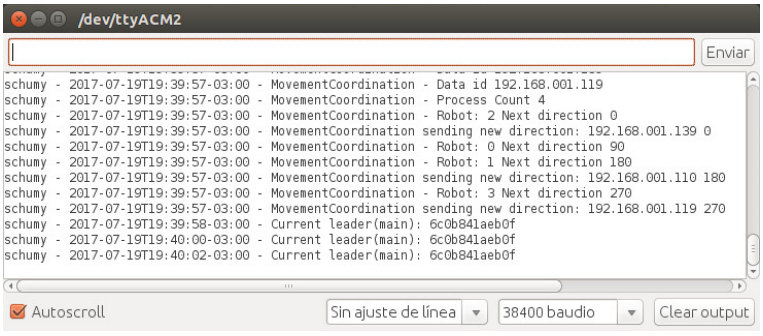
1. Al iniciar un dispositivo, siempre se debe considerar líder.
2. Luego del tiempo de estabilización, la red debe tener un solo líder.
3. Luego del tiempo de estabilización, todos los dispositivos deben considerar al mismo microcontrolador como líder.
4. Frente a la caída del líder, otro dispositivo debe tomar dicho rol, y dicho dispositivo debe ser único.
5. Si el dispositivo que era considerado líder, tiene una falla y abandona la red, y luego se recupera de la misma, debe tomar nuevamente su rol de líder.
6. Un dispositivo puede sumarse o abandonar la red en cualquier momento, y la misma debe seguir funcionando correctamente.
7. Si un dispositivo con mejor prioridad se suma a la red, este debe tomar el rol de líder.
8. Si un dispositivo con peor prioridad se suma a la red, este debe reconocer al líder de la misma.

3.1.2. Coordinación de movimiento

1. Solo el líder debe indicar las direcciones que deben tomar todos los dispositivos de la red.
2. Las direcciones que calcula el líder, deben garantizar que se cubra la mayor superficie.
3. Todos los dispositivos de la red, deben conocer la dirección del resto.
4. Si un dispositivo se suma o sale de la red, el líder debe recalcular las direcciones que deben tomar los demás.
5. A cada dirección se debe dirigir aquel dispositivo cuya dirección actual este más cerca.
6. Un dispositivo debe detenerse frente a la detección de un obstáculo.
7. Frente a la caída del líder, los dispositivos deben esperar que la red elija un nuevo líder, y este indique las nuevas direcciones.

A continuación se muestran registros obtenidos por el puerto serie del robot líder utilizados para la verificación de los casos antes mencionados. En el primer caso, se tienen 3 dispositivos en la red que deben seguir las direcciones 0°, 120°y 240°, mientras que en el segundo caso, se suma un cuarto robot, y por lo tanto, las nuevas direcciones son 0°, 90°, 180°y 270°.





3.2. Integración con Robot

El último paso, una vez probados los algoritmos, fue la integración en los robots. Las pruebas realizadas en este punto fueron íntegramente exploratorias. En un principio, se verificó el funcionamiento de un único robot, el cual debía dirigirse en una dirección específica. Una vez logrado el funcionamiento deseado, se montó toda la red de dispositivos IoT sobre los respectivos robots, y se hicieron pruebas visuales, verificando que todos los robots tomarán las direcciones esperadas. Se tuvieron en cuenta tanto las direcciones destino registradas de forma remota, como la dirección final que tomaron los robots.

Durante las pruebas integradoras de los microcontroladores y los robots, se descubrieron varios problemas. A continuación se enumeran los más relevantes:

1. El tiempo de respuesta de los sensores no había sido considerado durante el desarrollo, generando que el robot oscilara en su dirección, puesto que no se le dejaba el tiempo suficiente para medir la dirección actual.

2. El sensor ultrasónico no estaba correctamente alineado en algunos casos, provocando falsos positivos y negativos, detectando obstáculos donde no los había o no detectando obstáculos presentes.

3. El magnetómetro estaba siendo afectado por la interferencia de los motores y el chasis de aluminio del robot.

4. Los motores de los robots se alimentan independientemente, y por lo tanto, no se movían con la misma velocidad.

5. La superficie sobre la que se realizan las pruebas influye en el movimiento de los robots, principalmente en superficies donde las ruedas no presentan una buena adhesión.

4. Desarrollo

A continuación, se detalla la lista de tareas pensadas originalmente con su estimación, así como también las horas consumidas en cada tarea. En muchas de las tareas (principalmente de investigación y desarrollo de documentación), se utilizó el tiempo estimado como el tiempo máximo disponible para dicha tarea. Luego se detallan los problemas encontrados durante el desarrollo, y una breve explicación sobre los desvíos presentes.

4.1. Tareas

Tarea	Descripción	Horas Estimadas	Horas Consumidas
Confección de plan de proyecto	Creación de la propuesta, análisis y estimación de las tareas a desarrollar.	30	26
Gestión del proyecto	Seguimiento de las tareas y reuniones para coordinación del equipo.	96	96
Reuniones de avance	Reuniones para mostrar el avance que se realice durante el sprint. Se estima una reunión semanal de 1 hora de duración.	48	48
Configuración del ambiente de desarrollo	Instalación del entorno de desarrollo, configuración del repositorio de código.	24	28

Tarea	Descripción	Horas Estimadas	Horas Consumidas
Informe del trabajo	Confección del informe final del trabajo detallando los resultados obtenidos, así como también, los posibles futuros desarrollos, los inconvenientes encontrados y la documentación del sistema.	40	56
Tecnología			
Investigación de los dispositivos Photon	Investigación de los microcontroladores y realización de pruebas de concepto con los dispositivos.	75	90
Investigación del puente H	Investigación del funcionamiento y ensamblaje del puente H con motores de tensión continua.	40	55
Investigación del módulo HMC5883L	Investigación del módulo HMC5883L para ser utilizado como brújula digital.	100	110
Investigación del sensor de proximidad	Realizar pruebas de cómo se podría integrar un sensor de proximidad al microcontrolador para poder esquivar obstáculos.	35	30

Tarea	Descripción	Horas Estimadas	Horas Consumidas
Investigación del módulo ESP-8266	Investigar cómo generar una red AdHoc entre dos microcontroladores utilizando el módulo ESP-8266.	75	120
Algoritmo del líder			
Diseño del algoritmo de líder	Diseño de un algoritmo de elección de líder a partir del algoritmo Bully. Búsqueda de referencias y trabajos relacionados sobre algoritmos de elección de líder.	60	40
Desarrollo de un prototipo en un lenguaje de alto nivel	Implementación del algoritmo diseñado en un lenguaje de alto nivel para detectar de forma temprana cualquier caso no considerado.	30	30
Pruebas del prototipo	Pruebas sobre el prototipo para detectar problemas, utilizando herramientas de alto nivel (debugger, logs, etc).	32	32

Tarea	Descripción	Horas Estimadas	Horas Consumidas
Desarrollo del algoritmo en los microcontroladores	Implementación del algoritmo de líder en los microcontroladores y pruebas sobre los mismos.	80	100
Algoritmo de coordinación			
Diseño del algoritmo de coordinación	Diseño de un algoritmo de coordinación de movimiento por parte del líder, para que la red cubra la mayor superficie posible.	50	40
Desarrollo de un prototipo en un lenguaje de alto nivel	Implementación del algoritmo diseñado en un lenguaje de alto nivel para detectar de forma temprana cualquier caso no considerado.	50	50
Pruebas unitarias del prototipo	Pruebas sobre el prototipo para detectar problemas, utilizando herramientas de alto nivel (debugger, logs, etc).	32	32

Tarea	Descripción	Horas Estimadas	Horas Consumidas
Desarrollo del algoritmo en los microcontroladores	Implementación del algoritmo de coordinación en los microcontroladores y pruebas sobre los mismos.	180	250
Algoritmo de movimiento			
Diseño del algoritmo que gobierne el movimiento de los robots		25	20
Implementación del algoritmo de movimiento		60	80
Armado de robots			
Obtención de componentes	Compra de componentes necesarios. Plaquetas, cables, tablero de pruebas, baterías.	20	30
Ensamblado de chasis		10	15
Diseño del circuito		20	30
Soldado de plaquetas		25	30

4.1.1. De lo planificado a la realidad

Sobre un total de 1237 horas estimadas originalmente, se consumieron 1438 horas, siendo los mayores desvíos los presentados a la hora de implementar los algoritmos diseñados en lenguajes de alto nivel, en la plataforma provista por el microcontrolador.

Estos problemas se debieron en gran medida, a las limitaciones técnicas que presenta la plataforma, así como también la escasez de herramientas para la solución de problemas, tales como un mecanismo de debug. Por este motivo, se implementó como parte del trabajo, un sistema de registro por parte de los dispositivos, el cual puede leerse tanto por mensajes de red, como por el puerto serie del dispositivo.

También se detectaron serias falencias en la documentación de los microcontroladores en áreas cruciales para el desarrollo del trabajo, como lo son las comunicaciones o el multiprocesamiento (áreas que originalmente consideramos como una ventaja por parte de los dispositivos elegidos). Esto provocó que sea necesario analizar al código fuente del firmware para entender el funcionamiento de herramientas básicas como los sockets o los accesos a recursos compartidos. De esta forma, se encontró que los sockets presentan un comportamiento no bloqueante, diferente al esperado en la implementación del sistema operativo de una computadora. La imposibilidad de cambiar este comportamiento, obligó a la generación de esperas activas ("busy waits") los cuales verifican cada cierto tiempo si se de una condición determinada, en nuestro caso, si se recibieron nuevos mensajes. De no haberse recibido, el proceso queda en un estado inactivo por una cantidad arbitraria de tiempo y vuelve a verificar. Este comportamiento es indeseado, puesto que se consumen recursos del sistema sin realizar ninguna acción útil[28].

La mayor problemática surgió a la hora de integrar los diferentes algoritmos entre sí, junto con los sensores y actuadores (motores) reales. Puesto que la mayoría de los componentes de la aplicación fueron desarrollados por separado, los mismos se probaron utilizando valores fijos para las entradas y salidas de los algoritmos. A la hora de integrar todos los componentes, el comportamiento presentado por el microcontrolador fue errático, consumiendo una gran cantidad de horas dedicadas al análisis y corrección de errores, en su mayoría generados por lo expresado anteriormente respecto a la documentación y las limitaciones técnicas.

Por último, vale la pena destacar los problemas ocasionados por la electrónica misma, como lo son los tiempos de respuesta de los sensores, el manejo de la velocidad de los motores, o la interferencia generada al mantener la brújula muy cerca del chasis metálico del robot o los robots mismos.

[28] http://www.complang.tuwien.ac.at/scholz/publications/ada03.pdf

5. Conclusiones

Con el presente auge de los dispositivos IoT, se presentó una oportunidad única de poder realizar un trabajo que permita innovar en un área de la informática con un potencial, el cual está lejos de alcanzar un límite. En los tiempos que corren, escuchamos cada vez mas frases como teléfonos, electrodomésticos, autos inteligentes, para mencionar algunos, sin tener una noción certera de a que se refiere cuando se habla de un dispositivo inteligente.

En este trabajo se intentó dar una respuesta breve de a qué denominamos un dispositivo inteligente, y para dicha explicación integramos una red de dispositivos inteligentes para la solución de una problemática real, como lo es el barrido en zonas de desastre.

El trabajo combina el potencial de los dispositivos IoT con el de los sistemas distribuidos para lograr de esta forma, poder presentar un prototipo a escala de un sistema autónomo, capaz de recorrer una determinada superficie sin la interacción de un humano u otros dispositivos más que los robots mismos.

Es interesante remarcar que el sistema propuesto consta de dos componentes principales, por un lado están los robots en sí mismos, encargados del movimiento y el sensado del entorno (puede describirse como la parte **"mecánica" + hardware**) y por el otro, la implementación de los algoritmos descritos implementados en un microcontrolador (el **software**). Es decir, se podría utilizar cualquier otro mecanismo de movimiento y sensado de entorno, sin mayores cambios que los del hardware físico. Por ejemplo, si en el futuro se implementan los algoritmos presentados a gran escala, se podrían utilizar vehículos aéreos no tripulados (comúnmente denominados drones) para barrer superficies en las cuales el acceso terrestre es imposible o muy costoso (por ejemplo, para el caso mencionado anteriormente de la familia Pomar, o el velero Tunante II), o vehículos submarinos que faciliten búsquedas en lagos o ríos, entre otros (para el caso de la avioneta perdida en el Delta).

Como conclusión final del desarrollo, cabe destacar lo desafiante que fue aplicar dos áreas de estudio tan complejas pero con un gran potencial como lo son los sistemas distribuidos y los dispositivos IoT, si se lo utiliza con los fines adecuados.

6. Trabajos Futuros

Durante la realización del trabajo, encontramos funcionalidades que se considera deben realizarse en otra etapa del trabajo pues exceden el alcance del mismo:

- **Obstáculos**: Si bien el presente trabajo tuvo en cuenta los posibles obstáculos que se puedan encontrar los dispositivos en su recorrido,

no se implementó un algoritmo para sobreponerse a los mismos. Simplemente se procedió a detener el avance del dispositivo. Una posible mejora al sistema es agregando un algoritmo que intente esquivar los obstáculos. A continuación se propone una implementación posible de dicho algoritmo:

- Al encontrar un obstáculo, el dispositivo sigue los siguientes pasos.
    1. Espera un intervalo de tiempo aleatorio y vuelve a analizar si hay un obstáculo presente, si luego de varios ciclos, el dispositivo sigue detectando un obstáculo, entonces asumimos que el obstáculo no era temporal, y por lo tanto se procede al paso siguiente. En la presente implementación del sistema, el dispositivo se rige únicamente por este paso, si el obstáculo eventualmente se remueve, el robot vuelve a avanzar.
    2. Puesto que el obstáculo no es temporal, el dispositivo puede intentar esquivarlo. Para ello, puede elegir una dirección aleatoria cercana a la actual y seguir esa dirección por un intervalo corto de tiempo. Luego de ese intervalo, se vuelve a la dirección original y se analiza la presencia de obstáculos nuevamente. Si se vuelve a detectar un obstáculo se repite el proceso una cantidad determinada de veces. Si luego de dicha cantidad, el dispositivo sigue detectando un obstáculo, se asume que no hay forma de avanzar en dicha dirección y se detiene el dispositivo por completo.

- **Redes Ad Hoc**: Puesto que el sistema plantea una situación donde los dispositivos funcionen como una red autónoma, idealmente, no sería necesaria la comunicación a través de una red preestablecida. Es por ello que se plantea la generación de redes Ad Hoc, la cual permitirá a los controladores comunicarse sin necesidad dispositivos externos. Para lograr esto, se realizaron pruebas de conceptos utilizando dispositivos ESP-8266[29], los cuales permiten crear redes ad hoc por medio de comandos AT [30]. Sin embargo, la complejidad que presenta este desarrollo sumado a la duración máxima del trabajo, obligó a descartar dicha mejora.

- **Control de la velocidad**: Otra mejora posible que se le puede agregar al sistema, es la implementación de un PWM ("Modulación por anchos de pulsos"), el cual permite controlar la corriente que se le entrega a los motores de corriente continua, y de esta manera, controlar la velocidad de los mismos. Además de reducir considerablemente el consumo de energía (aumentando la vida útil de la batería), al reducir la velocidad del movimiento del robot, se le da más tiempo a los

sensores y actuadores de cumplir sus tareas.

- **RTOS**: Utilización de otro tipo de hardware que corra un sistema operativo de tiempo real.

7. Referencias

[1] Arun Kejariwal, Sanjeev Kulkarni, and Karthik Ramasamy. Real time analytics: Algorithms and systems. *PVLDB*, 8(12):2040–2041, 2015.

[2] Gang Liu Ben Zhang and Bi Hu. The coordination of nodes in the internet of things. *2010 International Conference on Information, Networking and Automation (ICINA)*, 2:V2–299–V2–302, 2010.

[3] Ahcène Bounceur, Madani Bezoui, Reinhardt Euler, Farid Lalem, and Massinissa Lounis. A Revised BROGO Algorithm for Leader Election in Wireless Sensor and IoT Networks. In *IEEE Sensors 2017*, IEEE Sensors 2017, Glasgow, United Kingdom, October 2017.

[4] Christoph Sommer, Florian Hagenauer, and Falko Dressler. A Networking Perspective on Self-Organizing Intersection Management. In *IEEE World Forum on Internet of Things (WF-IoT 2014)*, pages 230–234, Seoul, March 2014. IEEE.

[5] Yanjun Hu, Lei Zhang, Li Gao, and Enjie Ding. *Semi-Stochastic Topology Control with Application to Mobile Robot Team of Leader-Following Formation*, pages 125–134. Springer Berlin Heidelberg, 2015.

[6] Hector Garcia-Molina. Elections in a distributed computing system. *IEEE Trans. Computers*, 31, 1982.

[29] http://www.electroschematics.com/wp-content/uploads/2015/02/esp8266-datasheet.pdf

[30] https://www.sparkfun.com/datasheets/Cellular%20Modules/AT_Commands_Reference_Guide_r0.pdf

8. Código fuente

8.1. Prototipos en Java

```
1  package com.uba.fi.bully.status;
2
3  import java.util.Date;
4
5  public class BullyStatus {
6      private static BullyStatus instance= null;
7      private static final Object mutex= new Object();
8
9      public LeaderInfo info;
10
11     public static BullyStatus getInstance(){
12         if(instance==null){
13             synchronized (mutex){
14                 if(instance==null) instance = new
                       BullyStatus();
15             }
16         }
17         return instance;
18     }
19
20     private BullyStatus() {
21         this.info = new LeaderInfo();
22     }
23
24     public void setLeaderId(String newLeader) {
25         this.info.leaderId = newLeader;
26         this.info.timeStamp = new Date();
27     }
28 }
```

```
1  package com.uba.fi.data.status;
2
3  import java.util.Date;
4
5  public class CompassDirection {
6      public String id;
7      public Integer port;
8      public Double compassDirection;
9      public Date timeStamp;
10
11     CompassDirection(String id, Integer port, Double
           direction) {
12         this.id = id;
```

```
13          this.port = port;
14          this.compassDirection = direction;
15          this.timeStamp = new Date();
16      }
17
18  }
```

```
1   package com.uba.fi.data.worker;
2
3   import com.uba.fi.data.status.DataStatus;
4   import com.uba.fi.MulticastChannel;
5   import java.net.UnknownHostException;
6   import java.net.DatagramPacket;
7   import java.net.MulticastSocket;
8   import java.net.InetAddress;
9   import java.io.IOException;
10
11  public class CompassDirectionReceiver extends
        MulticastChannel {
12
13      public CompassDirectionReceiver(String id, String
            address, Integer port) {
14          super(id, address, port);
15      }
16
17      public void run() {
18          System.out.println("Process: " + this.id + "
                Compass Direction Receiver");
19
20          Boolean success = true;
21          InetAddress group = null;
22          MulticastSocket s = null;
23          try {
24              group = InetAddress.getByName(this.address);
25              s = new MulticastSocket(this.port);
26              s.joinGroup(group);
27          } catch (UnknownHostException e) {
28              e.printStackTrace();
29              success = false;
30          } catch (IOException e) {
31              e.printStackTrace();
32              success = false;
33          }
34
35          while (success) {
36              byte[] buf = new byte[1000];
37              DatagramPacket recv = new DatagramPacket(buf,
                    buf.length);
```

```
38              try {
39                  s.receive(recv);
40              } catch (IOException e) {
41                  e.printStackTrace();
42                  success = false;
43              }
44
45              String msg = new String(recv.getData(), recv.
                    getOffset(), recv.getLength());
46
47              String receivedId = msg.split("/")[0];
48              Integer receivedPort = Integer.parseInt(msg.
                    split("/")[1]);
49              Double receivedDirection = Double.parseDouble
                    (msg.split("/")[2]);
50
51              DataStatus.getInstance().setData(receivedId,
                    receivedPort, receivedDirection);
52          }
53
54          System.out.println("Compass Direction Receiver "
                + this.id + ": Quit unexpectedly");
55      }
56  }
```

```
1   package com.uba.fi.data.worker;
2
3   import com.uba.fi.data.status.DataStatus;
4   import com.uba.fi.util.DatagramSender;
5   import com.uba.fi.MulticastChannel;
6
7   public class CompassDirectionSender extends
        MulticastChannel {
8
9       private static final Integer DATA_TIMEOUT = 4;
10
11      public CompassDirectionSender(String id, String
            address, Integer port) {
12          super(id, address, port);
13      }
14
15      public void run() {
16          System.out.println("Process: " + this.id + "
                Compass Direction Sender running");
17
18          while (true) {
19              Double direction = DataStatus.getInstance().
                    getData(this.id).compassDirection;
```

```
20              Integer directionPort = DataStatus.
                    getInstance().getData(this.id).port;
21
22              String status = this.id + '/' + String.
                    valueOf(directionPort) + '/' + String.
                    valueOf(direction);
23              DatagramSender.sendMulticast(this.address,
                    this.port, status);
24
25              try {
26                  Thread.sleep(DATA_TIMEOUT * 1000);
27              } catch (InterruptedException e) {
28                  e.printStackTrace();
29              }
30          }
31      }
32  }
```

```
1   package com.uba.fi.data.status;
2
3   import java.util.Date;
4
5   public class DataStatus {
6
7       private static final Integer LIVENESS_TIMEOUT = 5;
8       private static final Integer ARRAY_SIZE = 10;
9
10      private static DataStatus instance= null;
11      private static final Object mutex= new Object();
12
13      private CompassDirection[] status;
14
15      public static DataStatus getInstance(){
16          if(instance==null){
17              synchronized (mutex){
18                  if(instance==null) instance= new
                        DataStatus();
19              }
20          }
21          return instance;
22      }
23
24      private DataStatus() {
25          this.status = new CompassDirection[ARRAY_SIZE];
26          for (int i = 0; i < ARRAY_SIZE; i++) {
27              CompassDirection emptyDirection = new
                    CompassDirection("", 0,-1.d);
28              status[i] = emptyDirection;
```

```
29          }
30      }
31
32      public void setData(String id, Integer port, Double
            direction) {
33          Boolean found = false;
34          for (int i = 0; i < ARRAY_SIZE; i++) {
35              // Set new data
36              if (id.equals(this.status[i].id)) {
37                  this.status[i] = new CompassDirection(id,
                        port, direction);
38                  found = true;
39              }
40              else if (! this.status[i].id.equals("")) {
41                  // Verify timeStamp and remove expired
                        values
42                  Date lastDate = this.status[i].timeStamp;
43                  long diffInMillies = (new Date()).getTime
                        () - lastDate.getTime();
44                  if (diffInMillies > (LIVENESS_TIMEOUT
                        *1000)) {
45                      this.status[i] = new CompassDirection
                            ("", 0, -1.d);
46                  }
47              }
48          }
49
50          // If the element was not found in the array, we
                should look for an empty space on it
51          if (!found) {
52              int i = 0;
53              while (i < ARRAY_SIZE && !this.status[i].id.
                    equals("")) {
54                  i++;
55              }
56
57              if (i < ARRAY_SIZE) {
58                  this.status[i] = new CompassDirection(id,
                        port, direction);
59              } else {
60                  // We couldn't find any empty space,
                        there are more proccess runing than
                        the available space
61                  System.err.println("Array overflow, there
                        are more proccesses runing than the
                        available space");
62              }
63          }
64      }
```

```
65
66      public CompassDirection getData(String id) {
67          CompassDirection value = null;
68          for (int i = 0; i < ARRAY_SIZE; i++) {
69              if (id.equals(this.status[i].id)) {
70                  value=this.status[i];
71              }
72          }
73          return value;
74      }
75
76      public  CompassDirection[] getData() {
77          return this.status;
78      }
79  }
```

```
1   package com.uba.fi.util;
2
3   import java.io.IOException;
4   import java.net.*;
5
6   public class DatagramSender {
7
8       public DatagramSender() {
9
10      }
11
12      public static void sendMulticast(String address,
            Integer port, String message) {
13          InetAddress group = null;
14          MulticastSocket s = null;
15          try {
16              group = InetAddress.getByName(address);
17              s = new MulticastSocket(port);
18              s.joinGroup(group);
19          } catch (UnknownHostException e) {
20              e.printStackTrace();
21          } catch (IOException e) {
22              e.printStackTrace();
23          }
24
25          DatagramPacket packet = new DatagramPacket(
                message.getBytes(), message.length(),
26                  group, port);
27          try {
28              s.send(packet);
29          } catch (IOException e) {
30              e.printStackTrace();
```

```
31          } catch (NullPointerException e) {
32              e.printStackTrace();
33          }
34      }
35
36      public static void sendDatagram(String address,
            Integer port, String message) {
37          InetAddress inetAddress = null;
38          DatagramSocket s = null;
39          try {
40              inetAddress = InetAddress.getByName(address);
41              s = new DatagramSocket();
42          } catch (UnknownHostException e) {
43              e.printStackTrace();
44          } catch (IOException e) {
45              e.printStackTrace();
46          }
47
48          DatagramPacket packet = new DatagramPacket(
                message.getBytes(), message.length(),
49                  inetAddress, port);
50          try {
51              s.send(packet);
52          } catch (IOException e) {
53              e.printStackTrace();
54          } catch (NullPointerException e) {
55              e.printStackTrace();
56          }
57      }
58  }
```

```
1   package com.uba.fi.movement;
2
3   import com.uba.fi.data.status.DataStatus;
4
5   import java.io.IOException;
6   import java.net.*;
7
8   public class DirectionReceiver extends Thread {
9
10      protected Integer port;
11      protected String id;
12
13      public DirectionReceiver(String id, Integer port) {
14          this.port = port;
15          this.id = id;
16      }
17
```

```
18      public void run() {
19          System.out.println("Process: " + this.id + " Next
                direction running");
20          Boolean success = true;
21          DatagramSocket s = null;
22          try {
23              s = new DatagramSocket(this.port);
24          } catch (IOException e) {
25              e.printStackTrace();
26              success = false;
27          }
28
29          while (success) {
30              byte[] buf = new byte[1000];
31              DatagramPacket recv = new DatagramPacket(buf,
                    buf.length);
32              try {
33                  s.receive(recv);
34              } catch (IOException e) {
35                  e.printStackTrace();
36                  success = false;
37              }
38
39              String msg = new String(recv.getData(), recv.
                   getOffset(), recv.getLength());
40              Double nextDirection = Double.parseDouble(msg
                   );
41
42              DataStatus.getInstance().setData(this.id,
                   this.port, nextDirection);
43          }
44          System.out.println("Next Direction Receiver " +
               this.id + ": Quit unexpectedly");
45      }
46  }
```

```
1   package com.uba.fi.bully.status;
2
3   import java.util.Date;
4
5   public class LeaderInfo {
6       /* We need to have all the leader info encapsulated
           to access it
7        * in a synchronized basis.
8        */
9       public String leaderId;
10      public Date timeStamp;
11  }
```

```java
package com.uba.fi.bully.worker;

import com.uba.fi.bully.status.BullyStatus;
import com.uba.fi.util.DatagramSender;
import com.uba.fi.MulticastChannel;

public class LeaderKeepAliveSender extends
    MulticastChannel {

    private static final Integer KEEP_ALIVE_TIMEOUT = 14;

    public LeaderKeepAliveSender(String id, String
        address, Integer port) {
        super(id, address, port);
    }

    public void run() {
        System.out.println("Process: " + this.id  + "
            Leader Keep Alive running");

        while (BullyStatus.getInstance().info.leaderId.
            equals(this.id)) {
            /* Every a given amount of time, if I'm the
                leader I should send a keep alive message
                .
             */
            DatagramSender.sendMulticast(this.address,
                this.port, this.id);

            try {
                Thread.sleep(KEEP_ALIVE_TIMEOUT*1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

```java
package com.uba.fi.bully.worker;

import com.uba.fi.bully.status.BullyStatus;
import com.uba.fi.bully.status.LeaderInfo;
import com.uba.fi.util.DatagramSender;

import java.util.Date;

public class LeaderLivenessVerifier extends Thread {
```

```
10
11      private static final Integer LIVENESS_TIMEOUT = 30;
            // In seconds
12
13      private String processId;
14      protected String address;
15      protected Integer port;
16
17      public LeaderLivenessVerifier(String id, String
            address, Integer port) {
18          this.processId = id;
19          this.address = address;
20          this.port = port;
21      }
22
23      public void run() {
24          System.out.println("Process: " + this.processId +
                " Leader Liveness Verifier running");
25
26          while (true) {
27              try {
28                  Thread.sleep(LIVENESS_TIMEOUT*1000);
29              } catch (InterruptedException e) {
30                  e.printStackTrace();
31              }
32
33              LeaderInfo info = BullyStatus.getInstance().
                    info;
34              if (! info.leaderId.equals(this.processId)) {
35                  Date lastDate = info.timeStamp;
36                  long diffInMillies = (new Date()).getTime
                        () - lastDate.getTime();
37
38                  if (diffInMillies > (LIVENESS_TIMEOUT
                        *1000)) {
39                      /* Last leader alive message was more
                             than LIVENESS_TIMEOUT seconds
                            ago, it's probably dead.
40                       * I proclaim myself as the new
                             leader.
41                       */
42                      BullyStatus.getInstance().setLeaderId
                            (this.processId);
43
44                      DatagramSender.sendMulticast(this.
                            address, this.port, this.
                            processId);
45                      (new LeaderKeepAliveSender(this.
                            processId, this.address, this.
```

```
                        port)).start();
46                  }
47              }
48          }
49
50      }
51  }
```

```
1   package com.uba.fi;
2
3   import com.uba.fi.data.worker.CompassDirectionReceiver;
4   import com.uba.fi.bully.worker.LeaderLivenessVerifier;
5   import com.uba.fi.data.worker.CompassDirectionSender;
6   import com.uba.fi.bully.worker.LeaderKeepAliveSender;
7   import com.uba.fi.movement.MovementCoordination;
8   import com.uba.fi.data.status.CompassDirection;
9   import com.uba.fi.movement.DirectionReceiver;
10  import com.uba.fi.bully.worker.ReceiverBully;
11  import com.uba.fi.bully.status.BullyStatus;
12  import com.uba.fi.data.status.DataStatus;
13  import java.util.Random;
14  import java.util.Date;
15
16  public class Main {
17
18      public static void main(String[] args) throws
            InterruptedException {
19
20          if (args.length != 5) {
21              System.err.println("Error: Invalid number of
                    arguments.\n Usage: ./script [Process Id]
                     [Bully Multicast Address] [Bully Data
                    Address] [Multicast Port] [Direction Port
                    ]");
22              System.exit(1);
23          }
24
25          String processId = args[0];
26          if (processId.length() != 12) {
27              System.err.println("Argument [Process Id]
                    must be 12 bytes long.");
28              System.exit(1);
29          }
30
31          String bullyAddress = args[1];
32          String dataAddress = args[2];
33
34          Integer port = 0;
```

```
35          try {
36              port = Integer.parseInt(args[3]);
37          } catch (NumberFormatException e) {
38              System.err.println("Argument [Multicast Port]
                    must be an integer.");
39              System.exit(1);
40          }
41
42          /* As we are running all the processes in the
               same computer, the way to
43           * receive messages from the leader are in
                localhost:[Direction Port],
44           * when the algorithm is implemented in the
                photons, we will use ip address.
45           */
46          Integer directionPort = 0;
47          try {
48              directionPort = Integer.parseInt(args[4]);
49          } catch (NumberFormatException e) {
50              System.err.println("Argument [Direction Port]
                    must be an integer.");
51              System.exit(1);
52          }
53
54          BullyStatus.getInstance().setLeaderId(processId);
55
56          Random randomGenerator = new Random();
57          Double randomDirection = randomGenerator.
               nextDouble();
58          DataStatus.getInstance().setData(processId,
               directionPort, randomDirection*360);
59
60          (new ReceiverBully(processId, bullyAddress, port)
               ).start();
61          (new LeaderKeepAliveSender(processId,
               bullyAddress, port)).start();
62          (new LeaderLivenessVerifier(processId,
               bullyAddress, port)).start();
63
64          (new CompassDirectionReceiver(processId,
               dataAddress, port)).start();
65          (new CompassDirectionSender(processId,
               dataAddress, port)).start();
66
67          (new DirectionReceiver(processId, directionPort))
               .start();
68          (new MovementCoordination(processId)).start();
69
70          while (true) {
```

```
71              Thread.sleep(30 * 1000);
72              // Only the leader should show this
                    informations
73              if (BullyStatus.getInstance().info.leaderId.
                    equals(processId)) {
74                  CompassDirection[] data = DataStatus.
                        getInstance().getData();
75
76                  for (CompassDirection aData : data) {
77                      if (aData.compassDirection != -1) {
78                          Date lastDate = aData.timeStamp;
79                          long diffInMillies = (new Date())
                                .getTime() - lastDate.getTime
                                ();
80                          String expired = (diffInMillies >
                                 (5 * 1000)) ? "Expired" : "
                                Valid";
81                          System.out.println("Process " +
                                aData.id + " receiving in
                                port "+ aData.port +" has
                                direction " + aData.
                                compassDirection + " and is "
                                 + expired);
82                      }
83                  }
84              }
85              else {
86                  CompassDirection myData = DataStatus.
                        getInstance().getData(processId);
87                  System.out.println("Process " + myData.id
                         + " receiving in port "+ myData.port
                         +" has direction " + myData.
                        compassDirection);
88              }
89          }
90      }
91  }
```

```
1   package com.uba.fi.command;
2
3   import com.uba.fi.util.DatagramSender;
4   import java.util.Scanner;
5
6   public class MainCommand {
7       public static void main(String[] args) {
8
9           if (args.length != 2) {
10              System.err.println("Error: Invalid number of
```

```
                arguments.\n Usage: ./script [Multicast
                Address] [Multicast Port]");
            System.exit(1);
        }


        String address = args[0];
        Integer port = 0;
        try {
            port = Integer.parseInt(args[1]);
        } catch (NumberFormatException e) {
            System.err.println("Argument [Multicast Port]
                must be an integer.");
            System.exit(1);
        }

        String availableCommands = "SsHhEe";
        char command = ' ';
        while(command != 'E') {
            command = ' ';
            while (availableCommands.indexOf(command) ==
                -1) {
                System.out.println("Enter command: 'S'
                    tart/'H'alt/'E'xit");
                Scanner reader = new Scanner(System.in);
                command = reader.next().charAt(0);
                command = Character.toUpperCase(command);
            }

            /* We currently aren't supporting Halt on the
                Photon */
            if (command == 'S') {
                DatagramSender.sendMulticast(address,
                    port, String.valueOf(command));
            }
        }
    }
}
```

```
package com.uba.fi.logger;

public class MainLogger {

    public static void main(String[] args) {

        if ((args.length < 2) || (args.length % 2 != 0))
            {
            System.err.println("Error: Invalid number of
```

```
                arguments.\n Usage: ./script [Multicast
                Address] [Multicast Port]");
            System.exit(1);
        }

        for (int i = 0; i < args.length/2; ++i) {
            String address = args[i*2];
            Integer port = 0;
            try {
                port = Integer.parseInt(args[i*2+1]);
            } catch (NumberFormatException e) {
                System.err.println("Argument [Multicast
                    Port] must be an integer.");
                System.exit(1);
            }

            MulticastLogger logger = new MulticastLogger(
                address, port);
            logger.start();
        }
    }
}
```

```
package com.uba.fi.monitor;

public class MainMonitor {

    public static void main(String[] args) {

        if ((args.length < 2) || (args.length % 2 != 0))
            {
            System.err.println("Error: Invalid number of
                arguments.\n Usage: ./script [Multicast
                Address] [Multicast Port]");
            System.exit(1);
        }

        for (int i = 0; i < args.length/2; ++i) {
            String address = args[i*2];
            Integer port = 0;
            try {
                port = Integer.parseInt(args[i*2+1]);
            } catch (NumberFormatException e) {
                System.err.println("Argument [Multicast
                    Port] must be an integer.");
                System.exit(1);
            }

```

```
22              MulticastMonitor monitor = new
                    MulticastMonitor(address, port);
23              monitor.start();
24          }
25      }
26  }
```

```
1   package com.uba.fi;
2
3   import java.io.Serializable;
4
5   public class Message implements Serializable {
6
7       private String processId;
8
9       public Message ( String s ) {
10          this.processId = s;
11      }
12
13      public String getProcessId () {
14          return this.processId;
15      }
16
17      public String toString() {
18          return this.processId;
19      }
20  }
```

```
1   package com.uba.fi.movement;
2
3   import com.uba.fi.data.status.CompassDirection;
4   import com.uba.fi.bully.status.BullyStatus;
5   import com.uba.fi.bully.status.LeaderInfo;
6   import com.uba.fi.data.status.DataStatus;
7   import com.uba.fi.util.DatagramSender;
8
9   import java.util.Date;
10
11  public class MovementCoordination extends Thread {
12
13      /* The amount of time that we should wait until we
            are sure that
14       * the bully algorithm is stable. And also we should
             have all the
15       * available robots directions.
16       */
17      private static final Integer STABILIZATION_TIME = 10;
```

```
18
19      /* We need to send the direction every given amount
           of seconds.
20       * We should consider that for this amount of time,
            each of the robots
21       * will follow the specify direction.
22       */
23      private static final Integer MOVEMENT_TIME = 3;
24
25
26      private String id;
27
28      public MovementCoordination(String id) {
29          this.id = id;
30      }
31
32      public void run() {
33          System.out.println("Process: " + this.id + "
               Movement Coordination running");
34
35          while (true) {
36              // Check if I am the current leader and if
                   the leader algorithm is already stable
37              LeaderInfo info = BullyStatus.getInstance().
                   info;
38
39              Date leaderSelectionTime = info.timeStamp;
40              long diffInMillies = (new Date()).getTime() -
                    leaderSelectionTime.getTime();
41              Boolean stable = (diffInMillies > (
                   STABILIZATION_TIME * 1000));
42
43              if (info.leaderId.equals(this.id) && stable)
                   {
44                  /* I have been the leader for a fair
                       amount of time, so we
45                   * can consider that the algorithm has
                        stabilized. I will send
46                   * each process its new direction.
47                   */
48                  this.sendDirectionInfo();
49              }
50
51              try {
52                  Thread.sleep(MOVEMENT_TIME*1000);
53              } catch (InterruptedException e) {
54                  e.printStackTrace();
55              }
56          }
```

```
57      }
58
59      /* In this method we check how many process do we
           have live,
60       * and send each of them a message with its new
            direction.
61       */
62      private void sendDirectionInfo() {
63          CompassDirection []directionInfo = DataStatus.
               getInstance().getData();
64
65          int arraySize = directionInfo.length;
66          Boolean freeRobot[] = new Boolean[arraySize];
67
68          int i = 0;
69          int processCount = 0;
70
71          for (i=0; i < arraySize; i++) {
72              if (directionInfo[i].compassDirection != -1)
                   {
73                  processCount++;
74                  freeRobot[i] = true;
75              }
76              else {
77                  freeRobot[i] = false;
78              }
79          }
80
81          for (int j=0; j < processCount; j++) {
82              Double nextDirection = j * 360.d/processCount
                   ;
83              int robotIndex = this.getClosestRobotIndex(
                   directionInfo, freeRobot, nextDirection);
84              freeRobot[robotIndex] = false;
85              this.sendProcessDirection(directionInfo[
                   robotIndex].port, nextDirection);
86              System.out.println("Robot: "+directionInfo[
                   robotIndex].id+ " should go: "+
                   nextDirection);
87          }
88      }
89
90      private int getClosestRobotIndex(CompassDirection []
           directionInfo, Boolean []freeRobot, Double
           nextDirection) {
91          int closestIndex = 0;
92          Double closestAngle = 640.d; // We choose an
               angle far enough
93          for (int i = 0; i < directionInfo.length; i++) {
```

```
94              if (freeRobot[i]) {
95                  CompassDirection direction =
                       directionInfo[i];
96                  Double difference = (java.lang.Math.abs(
                       nextDirection - direction.
                       compassDirection))%360;
97                  difference = difference > 180 ? 360 -
                       difference : difference;
98                  if (difference < closestAngle) {
99                      closestAngle = difference;
100                     closestIndex = i;
101                 }
102             }
103         }
104         return closestIndex;
105     }
106
107     private void sendProcessDirection(Integer port,
           Double direction) {
108         DatagramSender.sendDatagram("localhost", port,
               String.valueOf(direction));
109     }
110 }
```

```
1  package com.uba.fi;
2
3  public class MulticastChannel extends Thread {
4      protected String address;
5      protected Integer port;
6      protected String id;
7
8      public MulticastChannel(String id, String address,
           Integer port) {
9          this.address = address;
10         this.port = port;
11         this.id = id;
12     }
13 }
```

```
1  package com.uba.fi.logger;
2
3  import java.net.UnknownHostException;
4  import com.uba.fi.MulticastChannel;
5  import java.text.SimpleDateFormat;
6  import java.net.MulticastSocket;
7  import java.net.DatagramPacket;
8  import java.io.BufferedWriter;
```

```
9  import java.net.InetAddress;
10 import java.io.IOException;
11 import java.io.FileWriter;
12 import java.util.Calendar;
13 import java.io.File;
14
15 public class MulticastLogger extends MulticastChannel {
16
17     MulticastLogger(String address, Integer port) {
18         super("", address, port);
19     }
20
21     public void run() {
22         System.out.println("Receiver "+this.address + "
               running");
23
24         Boolean success = true;
25
26         MulticastSocket s = null;
27         InetAddress group = null;
28         try {
29             group = InetAddress.getByName(this.address);
30             s = new MulticastSocket(this.port);
31             s.joinGroup(group);
32         } catch (UnknownHostException e) {
33             e.printStackTrace();
34             success = false;
35         } catch (IOException e) {
36             e.printStackTrace();
37             success = false;
38         }
39
40         String timeLog = "Log";
41         File f = new File(timeLog);
42         if(f.exists() && !f.isDirectory()) {
43             String timeStamp = new SimpleDateFormat("
                   yyyyMMdd_HH:mm:ss").format(Calendar.
                   getInstance().getTime());
44             f.renameTo(new File(timeLog+"-"+timeStamp));
45         }
46
47         BufferedWriter writer = null;
48         try {
49             // Create a temporary file
50
51             File logFile = new File(timeLog);
52             // This will output the full path where the
                   file will be written to...
53             System.out.println(logFile.getCanonicalPath()
```

```
                );
54              writer = new BufferedWriter(new FileWriter(
                    logFile));
55          } catch (IOException e1) {
56              e1.printStackTrace();
57          }
58
59          while (success) {
60              // Join a Multicast group
61
62              byte[] buf = new byte[1000];
63              DatagramPacket recv = new DatagramPacket(buf,
                    buf.length);
64              try {
65                  s.receive(recv);
66              } catch (IOException e) {
67                  e.printStackTrace();
68                  success = false;
69              }
70
71              String msg = new String(recv.getData(), recv.
                    getOffset(), recv.getLength());
72
73              System.out.println("Receiver " + this.address
                    + ": "+ msg);
74
75              try {
76                  writer.write(msg);
77                  writer.newLine();
78                  writer.flush();
79              } catch (IOException e) {
80                  e.printStackTrace();
81              }
82          }
83
84          System.out.println("Receiver " + this.address + "
                : Quit undetectably");
85          try {
86              s.leaveGroup(group);
87          } catch (IOException e) {
88              e.printStackTrace();
89          }
90      }
91  }
```

```
1   package com.uba.fi.monitor;
2
3   import java.net.UnknownHostException;
```

```
4  import com.uba.fi.MulticastChannel;
5  import java.net.MulticastSocket;
6  import java.net.DatagramPacket;
7  import java.net.InetAddress;
8  import java.io.IOException;
9
10 public class MulticastMonitor extends MulticastChannel {
11
12     MulticastMonitor(String address, Integer port) {
13         super("", address, port);
14     }
15
16     public void run() {
17         System.out.println("Receiver "+this.address + ":"
               +port + " running");
18
19         Boolean success = true;
20
21         MulticastSocket s = null;
22         InetAddress group = null;
23         try {
24             group = InetAddress.getByName(this.address);
25             s = new MulticastSocket(this.port);
26             s.joinGroup(group);
27         } catch (UnknownHostException e) {
28             e.printStackTrace();
29             success = false;
30         } catch (IOException e) {
31             e.printStackTrace();
32             success = false;
33         }
34
35         while (success) {
36             // Join a Multicast group
37
38             byte[] buf = new byte[1000];
39             DatagramPacket recv = new DatagramPacket(buf,
                   buf.length);
40             try {
41                 s.receive(recv);
42             } catch (IOException e) {
43                 e.printStackTrace();
44                 success = false;
45             }
46
47             String msg = new String(recv.getData(), recv.
                  getOffset(), recv.getLength());
48             System.out.println("Receiver " + this.address
                   + ":"+port+ " - "+ msg);
```

```
49          }
50
51          System.out.println("Receiver " + this.address + "
                : Quit undetectably");
52          try {
53              s.leaveGroup(group);
54          } catch (IOException e) {
55              e.printStackTrace();
56          }
57      }
58  }
```

```
1   package com.uba.fi.bully.worker;
2
3   import com.uba.fi.bully.status.BullyStatus;
4   import java.net.UnknownHostException;
5   import com.uba.fi.MulticastChannel;
6   import java.net.MulticastSocket;
7   import java.net.DatagramPacket;
8   import java.net.InetAddress;
9   import java.io.IOException;
10  import com.uba.fi.Message;
11
12  public class ReceiverBully extends MulticastChannel {
13
14      public ReceiverBully(String id, String address,
            Integer port) {
15          super(id, address, port);
16      }
17
18      public void run() {
19          System.out.println("Process: " + this.id + "
                Receiver Bully running");
20
21          Boolean success = true;
22          InetAddress group = null;
23          MulticastSocket s = null;
24          try {
25              group = InetAddress.getByName(this.address);
26              s = new MulticastSocket(this.port);
27              s.joinGroup(group);
28          } catch (UnknownHostException e) {
29              e.printStackTrace();
30              success = false;
31          } catch (IOException e) {
32              e.printStackTrace();
33              success = false;
34          }
```

```
35
36          while (success) {
37              byte[] buf = new byte[1000];
38              DatagramPacket recv = new DatagramPacket(buf,
                    buf.length);
39              try {
40                  s.receive(recv);
41              } catch (IOException e) {
42                  e.printStackTrace();
43                  success = false;
44              }
45
46              String msg = new String(recv.getData(), recv.
                   getOffset(), recv.getLength());
47              Message receivedMessage = new Message(msg);
48
49              if (! receivedMessage.getProcessId().equals(
                   this.id)) {
50                  String currentLeader = BullyStatus.
                       getInstance().info.leaderId;
51                  int result = currentLeader.compareTo(
                       receivedMessage.getProcessId());
52                  /* result > 0: New leader was found (
                       currentLeader < receivedMessage.
                       getProcessId())
53                   * result < 0: Someone proclaims to be a
                        leader, but is not
54                   * result = 0: The current leader is
                        alive
55                   */
56                  if (result < 0) {
57                      // Someone has a wrong leader
58                      if (currentLeader.equals(this.id)) {
59                          // I'm the leader, so I should
                               tell everyone
60                          DatagramPacket leader = new
                               DatagramPacket(this.id.
                               getBytes(), this.id.length(),
61                                    group, this.port);
62                          try {
63                              s.send(leader);
64                          } catch (IOException e) {
65                              e.printStackTrace();
66                              success = false;
67                          }
68                      }
69                  } else if (result > 0) {
70                      // New Leader
71                      BullyStatus.getInstance().setLeaderId
```

```
                        (receivedMessage.getProcessId());
72                  } else {
73                      // Leader Alive
74                      BullyStatus.getInstance().setLeaderId
                            (currentLeader); // Timestamp is
                            reseted
75                  }
76              }
77          }
78
79          System.out.println("Receiver " + this.id + ":
                Quit undetectably");
80          try {
81              s.leaveGroup(group);
82          } catch (IOException e) {
83              e.printStackTrace();
84          }
85      }
86  }
```

8.2. Photon

```
1   #include "AddressManager.h"
2
3   /* Returns the ip address of the device formatted as
4      a String
5    */
6    String AddressManager::getIp(){
7       IPAddress addr = WiFi.localIP();
8
9       char buf[4];
10
11      sniprintf(buf, 4, "%03d", addr[0]);
12      String result = String(buf);
13
14      for (size_t i = 1; i <= 3; i++) {
15        result += ".";
16        sniprintf(buf, 4, "%03d", addr[i]);
17        result += String(buf);
18      }
19
20      return result;
21  }
22
23  /* Returns the mac address of the device formatted as
24     a String
25   */
```

```
26 String AddressManager::getMac(){
27     byte mac[6];
28     WiFi.macAddress(mac);
29 
30     char macString[18];
31     sniprintf(macString, 18, "%02x%02x%02x%02x%02x%02x",
           mac[0], mac[1], mac[2], mac[3], mac[4], mac[5]);
32 
33     String result(macString);
34     return result;
35 }
36 
37 /* Transform a string ip address to a byte array
38  */
39 void AddressManager::toIp(String ip, byte ipReturn[]) {
40     int init = 0;
41     int fin = 0;
42 
43     for (size_t i = 0; i < 4; i++) {
44         fin = ip.indexOf(".", init);
45         ipReturn[i] = ip.substring(init, fin).toInt();
46 
47         init = fin + 1;
48     }
49 }
```

```
1 
2 #ifndef ADDRESS_MANAGER_H
3 #define ADDRESS_MANAGER_H
4 
5     #include "Constants.h"
6 
7     class AddressManager{
8         public:
9 
10             // Ip/mac address info
11             static String getIp();
12             static String getMac();
13 
14             static void toIp(String ip, byte ipReturn[]);
15     };
16 
17 #endif
```

```
1 #include "Movement/MovementDirectionReceiver.h"
2 #include "AdafruitSensor/Adafruit_HMC5883_U.h"
3 #include "CompassSensor/DirectionsManager.h"
```

```
4  #include "ProximitySensor/ProximitySensor.h"
5  #include "AdafruitSensor/Adafruit_Sensor.h"
6  #include "Compass/CompassDirectionSender.h"
7  #include "AddressManager/AddressManager.h"
8  #include "Movement/MovementCoordination.h"
9  #include "Bully/LeaderLivenessVerifier.h"
10 #include "Compass/DataCompassDirection.h"
11 #include "Bully/KeepAliveSender.h"
12 #include "CompassSensor/Compass.h"
13 #include "Bully/ReceiverBully.h"
14 #include "Compass/DataStatus.h"
15 #include "Bully/BullyStatus.h"
16 #include "Logger/OnroLogger.h"
17 #include "Constants.h"
18
19 BullyStatus *bullyStatus;
20
21 ProximitySensor *proximitySensor;
22 DirectionsManager *directionManager;
23
24 int lastCheckForObstacles;
25 bool nearObstacle;
26
27 int currentDirection;
28 int destinationDirection;
29
30 char command;
31
32 int i = 0;
33
34 void setupInitialData() {
35   // Bully algorithm
36   String myId = AddressManager::getMac();
37   bullyStatus = new BullyStatus(myId);
38
39   // Compass direction algorithm
40   currentDirection = 0;
41   destinationDirection = 0;
42
43   // Near obstacles algorithm
44   lastCheckForObstacles = 0;
45   nearObstacle = true;
46
47   // Start command
48   command = COMMAND_HOLD;
49 }
50
51 // setup() runs once, when the device is first turned on.
52 void setup() {
```

```
53
54    waitUntil(WiFi.ready);
55
56    // Init the Logger
57    OnroLogger::getInstance();
58    Particle.subscribe("spark/", nameHandler);
59    Particle.publish("spark/device/name");
60
61    setupInitialData();
62
63    pinMode(LEADER_PIN, OUTPUT);
64
65    directionManager = new DirectionsManager(MOTOR_1_PIN,
          MOTOR_2_PIN, COMPASS_TOLERANCE, declinationAngle);
66    proximitySensor = new ProximitySensor(ECHO_PIN,
          TRIGGER_PIN);
67
68    KeepAliveSender keepAlivesender(bullyStatus);
69    keepAlivesender.start();
70
71    LeaderLivenessVerifier verifier(bullyStatus);
72    verifier.start();
73
74    ReceiverBully receiverBully(bullyStatus);
75    receiverBully.start();
76  }
77
78  void nameHandler(const char *topic, const char *data) {
79    OnroLogger::getInstance().setDeviceName(String(data));
80  }
81
82  // loop() runs over and over again, as quickly as it can
       execute.
83  void loop() {
84
85    if (command != COMMAND_START) {
86      OnroLogger::getInstance().logln("Main - Waiting start
            command");
87
88      digitalWrite(LEADER_PIN, LOW);
89
90      UDP commandUdp;
91      commandUdp.begin(commandPort);
92      commandUdp.joinMulticast(groupIp);
93
94      while (COMMAND_START != command) {
95        delay(250);
96
97        turnLedIfLeader();
```

```
98
99        if (commandUdp.parsePacket() > 0) {
100         command = commandUdp.read();
101       }
102       if (command == COMMAND_START) {
103         // We receive the Start command, so we launch the
              movement coordination algorithm
104         OnroLogger::getInstance().logln("Main - Start
              command receive");
105
106         double initialAngle = getInitialAngle();
107         OnroLogger::getInstance().logln("Main - Initial
              Angle: "+String(initialAngle));
108         SINGLE_THREADED_BLOCK() {
109           currentDirection = initialAngle;
110         }
111
112         nearObstacle = checkForObstacles();
113         lastCheckForObstacles = Time.now();
114
115         // Need my current direction to send
116         CompassDirectionSender sender(&currentDirection);
117         sender.start();
118
119         // Receive the direction where I should move
120         MovementDirectionReceiver movementReceiver(&
              destinationDirection);
121         movementReceiver.start();
122
123         // Receive all the directions from the other
              robots and send the destination to each robot
124         MovementCoordination movementCoordination(
              bullyStatus, &currentDirection, &
              destinationDirection);
125         movementCoordination.start();
126
127         commandUdp.leaveMulticast(groupIp);
128       }
129     }
130   }
131
132   if (shouldCheckForObstacles()) {
133     lastCheckForObstacles = Time.now();
134     nearObstacle = checkForObstacles();
135   }
136
137   if (bullyStatus->isStable()) {
138     if (nearObstacle) {
139       directionManager->disable();
```

```
140       }
141       else {
142         if (i%2 == 0) {
143           directionManager->disable();
144         }
145         else {
146           directionManager->enable();
147         }
148         i++;
149       }
150     }
151     else {
152       directionManager->disable();
153     }
154
155     int currentAngle = String(directionManager->
          getCurrentDir()).toInt();
156     int targetAngle = 0;
157
158     SINGLE_THREADED_BLOCK() {
159       currentDirection = currentAngle;
160       targetAngle = destinationDirection;
161     }
162
163
164     directionManager->headFoward(targetAngle);
165
166     OnroLogger::getInstance().logln("Main - Current Angle:
          "+String(currentAngle) + " Target Angle "+String(
          targetAngle) );
167
168     turnLedIfLeader();
169
170     uint32_t freemem = System.freeMemory();
171     OnroLogger::getInstance().logln("Main - Free memory: "+
          String(freemem));
172
173     delay(500);
174   }
175
176   bool checkForObstacles() {
177     long distance = proximitySensor->getDistance();
178     OnroLogger::getInstance().logln("Main - Distance: "+
          String(distance));
179
180     return (distance > 0) && (distance < SENSOR_TOLERANCE);
181   }
182
183   bool shouldCheckForObstacles() {
```

```
184   int timeDifference = Time.now() - lastCheckForObstacles
          ;
185   return (timeDifference > INTERVAL_TIMER);
186 }
187
188 double getInitialAngle() {
189   double acumulativeAngle = 0;
190   int numberOfSamples = 3;
191   for (int i = 0; i < numberOfSamples; ++i) {
192     acumulativeAngle += directionManager->getCurrentDir()
          ;
193     delay(INITIAL_ANGLE_TIME*1000);
194   }
195   return acumulativeAngle/numberOfSamples;
196 }
197
198 void turnLedIfLeader() {
199   String currentLeader = bullyStatus->getLeaderId();
200
201   if (AddressManager::getMac().equals(currentLeader)) {
202     digitalWrite(LEADER_PIN, HIGH);
203   }
204   else {
205     digitalWrite(LEADER_PIN, LOW);
206   }
207 }
208
209 /*
     ************************************************************************


210
211 Pin configuration:
212
213 Compass     Photon
214 5v      ->  -----
215 Gnd     ->  Gnd
216 SCL     ->  pin D1
217 SDA     ->  pin D0
218 drdy    ->  -----
219 3v3     ->  3v3
220
221 Sensor      Photon
222 5v      ->  5v
223 Gnd     ->  Gnd
224 TRIG    ->  A0
225 ECHO    ->  A1
226
227 H-Bridge    Photon
228 5v       -> 5v
```

```
Gnd       -> Gnd
in1(rev) -> -----
in1            -> pin D5
in4       -> pin D4
in4(rev) -> -----

 ********************************************************************************
    */
```

```
#include "BullyStatus.h"

#include "Logger/OnroLogger.h"
#include "AddressManager/AddressManager.h"

BullyStatus::BullyStatus(String leaderId) {
   this->asynSetLeader(leaderId);
}

void BullyStatus::asynSetLeader(String newLeader) {
    if (! this->leaderId.equals(newLeader)) {
      this->leaderUpdateTimeStamp = Time.now();
    }
    this->leaderId = newLeader;
    this->timeStamp = Time.now();
}

void BullyStatus::setLeaderId(String newLeader) {
    SINGLE_THREADED_BLOCK() {
        this->asynSetLeader(newLeader);
    }
}

String BullyStatus::getLeaderId() {
     SINGLE_THREADED_BLOCK() {
        return this->leaderId;
    }
    OnroLogger::getInstance().logln("It shouldn't enter
       here");
    return "";
}

int BullyStatus::getTimeStamp() {
     SINGLE_THREADED_BLOCK() {
        return this->timeStamp;
    }
    OnroLogger::getInstance().logln("It shouldn't enter
       here");
    return -1;
```

```
38 }
39
40 bool BullyStatus::isStable() {
41
42   bool stable = false;
43   SINGLE_THREADED_BLOCK() {
44     int currentTime = Time.now();
45     int timeDifference = currentTime - this->
          leaderUpdateTimeStamp;
46     stable = (timeDifference > STABILIZATION_TIME);
47   }
48   return  stable;
49 }
```

```
1
2 #ifndef BULLY_STATUS_H
3 #define BULLY_STATUS_H
4
5 #include "application.h"
6
7     class BullyStatus {
8
9         private:
10            String leaderId;
11
12            /* The moment where a message from the leader
                 is received.
13             * If this value is not updated for a given
                 amount of timeStamp
14             * (LIVENESS_TIMEOUT) it's used to detect if
                 the leader has fallen.
15             */
16            int timeStamp;
17
18            /* The moment where a different leader is
                 selected.
19             * If this value is to soon, the algorithm
                 that selects the
20             * leader, is not stable yet.
21             */
22            int leaderUpdateTimeStamp;
23
24            void asynSetLeader(String newLeader);
25
26        public:
27
28            BullyStatus (String leaderId);
29            void setLeaderId(String newLeader);
```

```
30
31              String getLeaderId();
32              int getTimeStamp();
33              bool isStable();
34      };
35
36  #endif
```

```
1   #include "math.h"
2   #include "Compass.h"
3   #include "AdafruitSensor/Adafruit_Sensor.h"
4
5   Compass::Compass(float declinationAngle) {
6       this->declination = declinationAngle;
7       this->mag = Adafruit_HMC5883_Unified(12345);
8       mag.begin();
9   }
10
11  float Compass::getAngle() {
12
13    const double pi = 3.14159265358979323846264338327 95;
14
15    /* Get a new sensor event */
16    sensors_event_t event;
17    this->mag.getEvent(&event);
18
19    float heading = atan2(event.magnetic.y, event.magnetic.
          x);
20    heading +=  this->declination;
21
22    // Correction for when signs are reversed.
23    if(heading < 0)
24      heading += 2*pi;
25
26    // Check for wrap due to addition of declination.
27    if(heading > 2*pi)
28      heading -= 2*pi;
29
30    // Convert radians to degrees for readability.
31    float headingDegrees = heading * 180/pi;
32    return headingDegrees;
33  }
```

```
1   #ifndef COMPASS_H
2   #define COMPASS_H
3
4     #include "application.h"
```

```
5     #include "AdafruitSensor/Adafruit_HMC5883_U.h"
6
7     class Compass{
8       public:
9         Compass(float declination);
10        float getAngle();
11
12      private:
13        float declination;
14        Adafruit_HMC5883_Unified mag;
15      };
16
17  #endif
```

```
1   #include "AddressManager/AddressManager.h"
2   #include "CompassDirectionSender.h"
3   #include "Logger/OnroLogger.h"
4   #include "Constants.h"
5   #include "application.h"
6
7   Thread *compassDirectionSenderThread;
8
9   os_thread_return_t compassDirectionSender(void* param){
10
11      int *currentDirection = (int *)param;
12
13      String myId = AddressManager::getIp();
14
15      while (true) {
16          UDP sendUdp;
17          sendUdp.begin(groupPortCD+1);
18
19          int direction = 0;
20          SINGLE_THREADED_BLOCK() {
21            direction = *currentDirection;
22          }
23          CompassDirection dataStruct = { myId, direction
                };
24
25          String dataToSend = DataCompassDirection::
                toString(dataStruct);
26
27          OnroLogger::getInstance().logln("
                CompassDirectionSender - Data to send "+
                dataToSend);
28
29          char buffer[dataLength];
30          dataToSend.toCharArray(buffer, dataLength);
```

```
31
32          IPAddress remoteIP(groupIp);
33          int port = groupPortCD;
34          sendUdp.beginPacket(remoteIP, port);
35          sendUdp.write(reinterpret_cast<const unsigned
               char*>(buffer), dataLength);
36          sendUdp.endPacket();
37
38          delay(PUBLISH_TIME*1000);
39      }
40  }
41
42  CompassDirectionSender::CompassDirectionSender (int *
       currentDirection) {
43      this->currentDirection = currentDirection;
44  }
45
46  void CompassDirectionSender::start() {
47      compassDirectionSenderThread = new Thread("
           CompassDirectionSender", compassDirectionSender,
           this->currentDirection);
48  }
```

```
1   #ifndef COMPASS_DIRECTION_SENDER_H
2   #define COMPASS_DIRECTION_SENDER_H
3
4   #include "DataStatus.h"
5
6   class CompassDirectionSender {
7
8     private:
9         int *currentDirection;
10
11    public:
12        CompassDirectionSender (int *currentDirection);
13        void start();
14  };
15
16  #endif
```

```
1   #ifndef MY_CONSTS
2   #define MY_CONSTS
3
4     #include "application.h"
5
6     const int maxCount = 5; // Maximun amount of robots
         that can join the network
```

```
7
8     const int idLength = 13; // Length of the id we are
          using for the bully algorithm
9
10    const int ipLength = 15; // Length of an ip address as
          string 127.000.000.001
11    const int compassDirectionLength = 4;
12    const int dataLength = ipLength +
          compassDirectionLength;
13
14    const byte groupIp[] = { 224,22,4,89 }; // Multicast
          address use to publish my ip
15    const int groupPort = 50005; // Multicast port use to
          publish my ip
16    const int groupPortCD = 60006; // Multicast port use to
           publish my compass direction
17    const int portMovement = 40004; // Unicast port use to
          receive the direction where I should go
18    const int commandPort = 30003; // Multicast port use to
           receive the 'Start' command
19
20    const byte logIp[] = { 224,22,4,88 }; // Multicast
          address use to send logs
21    const int logPort = 62000; // Multicast port use to
          send logs
22
23    const int baudRate = 9600; // Baud rate used to
          communicate via Serial
24
25    const float declinationAngle = 0.1488766; //
          Declination used to calibrate the compass according
           the earth positioning
26
27    /* The amount of time after which we consider the
          leader has fallen
28     * if we don't receive a keep alive message from him.
29     */
30    const int LIVENESS_TIMEOUT = 5;
31
32    /* The amount of time every which the leader sends a
          keep alive
33     * message.
34     */
35    const int KEEP_ALIVE_TIMEOUT = 2;
36
37    /* The amount of time that we should wait until we are
          sure that
38     * the bully algorithm is stable. And also we should
           have all the
```

```
39      * available robots directions.
40      */
41     const int STABILIZATION_TIME = 10;
42
43     /* We need to send the direction every given amount of
           seconds.
44      * We should consider that for this amount of time,
            each of the robots
45      * will follow the specify direction.
46      */
47     const int MOVEMENT_TIME = 2;
48
49     /* The time interval between compass direction
           pubblishes.
50      * Used by CompassDirectionSender
51      */
52     const int PUBLISH_TIME = 2;
53
54     /* The time interval between proximity sensor measures
           */
55     const int INTERVAL_TIMER = 1;
56
57     /* To have an accurate measurement of the angle, the
           first time,
58      * we measure the angle several times every
            INITIAL_ANGLE_TIME
59      * seconds
60      */
61     const double INITIAL_ANGLE_TIME = 0.5;
62
63     /* The tolerance to consider when positioning the robot
            in
64      * the target angle.
65      */
66     const int COMPASS_TOLERANCE = 50;
67
68     /* The tolerance to consider when detecting near
           obstacles */
69     const int SENSOR_TOLERANCE = 30;
70
71     /* Pin values */
72     const int TRIGGER_PIN = A0;
73     const int ECHO_PIN = A1;
74     const int MOTOR_1_PIN = D5;
75     const int MOTOR_2_PIN = D4;
76     const int LEADER_PIN = D7;
77
78     /* Commands received to start the movement algorithm */
79     const char COMMAND_HOLD = 'H'; // Currently unused
```

```
80    const char COMMAND_START = 'S';
81
82  #endif
```

```
1   #include "DataCompassDirection.h"
2
3   // Return Photon's compass direction data
4   CompassDirection DataCompassDirection::toCompassDirection
        (String data) {
5       String id = data.substring(0, ipLength);
6       String direction = data.substring(ipLength,
            dataLength);
7
8       CompassDirection result = { id, direction.toInt() };
9       return result;
10  }
11
12  String DataCompassDirection::toString(CompassDirection
        data){
13      char bufDir[compassDirectionLength + 1];
14      sniprintf(bufDir, compassDirectionLength + 1, "%03d",
            data.compassDirection);
15      String directionString = String(bufDir);
16
17      return data.id + directionString;
18  }
```

```
1   #ifndef DATA_COMPASS_DIRECTION_H
2   #define DATA_COMPASS_DIRECTION_H
3
4   #include "Constants.h"
5
6   struct CompassDirection {
7       String id;
8       int compassDirection;
9   };
10
11  class DataCompassDirection {
12    public:
13        static CompassDirection toCompassDirection(String
              data);
14        static String toString(CompassDirection data);
15  };
16
17  #endif
```

```
#include "DataStatus.h"

#include "Logger/OnroLogger.h"

DataStatus::DataStatus(CompassDirection myData) {
  for (size_t i = 0; i < maxCount; i++) {
    this->dataSubjects[i] = { {"", -1}, -1};
  }
  this->asynSetData(myData);
}

void DataStatus::asynSetData(CompassDirection newData) {
  int currentTime = Time.now();
  bool found = false;

  for (size_t i = 0; i < maxCount; i++) {
    // Set new data
    if (newData.id.equals(this->dataSubjects[i].direction
        .id)) {
      // The process is already in our array, so we must
         update its direction
      this->dataSubjects[i].direction.compassDirection =
         newData.compassDirection;
      this->dataSubjects[i].timeStamp = currentTime;
      found = true;
    }
  }

  if (!found) {
    // We need to add a new element to the array instead
       of updating one.
    size_t i = 0;
    // Look for the first free element of the array
    while (i < arraySize(this->dataSubjects) && !this->
       dataSubjects[i].direction.id.equals("")) {
      i++;
    }

    if (i < maxCount) {
      this->dataSubjects[i] = { newData, currentTime };
    } else {
      OnroLogger::getInstance().logln("Array OVERFLOW!!!!
         ");
    }
  }
}

void DataStatus::cleanExpired() {
```

```
43    int currentTime = Time.now();
44    for (size_t i = 0; i < maxCount; i++) {
45      if (!this->dataSubjects[i].direction.id.equals("")) {
46        // Verify timeStamp to remove all those element
              that are expired
47        int timeDifference = currentTime - this->
              dataSubjects[i].timeStamp;
48
49        if (timeDifference > (LIVENESS_TIMEOUT)) {
50          OnroLogger::getInstance().logln("Removing: "+
                String(this->dataSubjects[i].direction.id));
51          this->dataSubjects[i] = { {"", -1}, -1};
52        }
53      }
54    }
55  }
56
57  void DataStatus::setData(CompassDirection newData) {
58    this->asynSetData(newData);
59  }
60
61  CompassDirectionWithTimeStamp DataStatus::getDataFrom(
       String id) {
62    CompassDirectionWithTimeStamp returnValue = { {"", -1},
          -1};
63    for (size_t i = 0; i < arraySize(this->dataSubjects); i
        ++) {
64      if (id.equals(this->dataSubjects[i].direction.id)) {
65        returnValue = this->dataSubjects[i];
66        break;
67      }
68    }
69    return returnValue;
70  }
71
72  CompassDirectionWithTimeStamp DataStatus::
       getDataFromIndex(size_t index) {
73    return this->dataSubjects[index];
74  }
```

```
1   #ifndef DATA_STATUS_H
2   #define DATA_STATUS_H
3
4   #include "DataCompassDirection.h"
5   #include "application.h"
6   #include "Constants.h"
7
8   struct CompassDirectionWithTimeStamp {
```

```
9     CompassDirection direction;
10    int timeStamp;
11  };
12
13  class DataStatus {
14
15    private:
16
17      CompassDirectionWithTimeStamp dataSubjects[maxCount]
            = {};
18      void asynSetData(CompassDirection newData);
19
20    public:
21      DataStatus(CompassDirection myData);
22      void setData(CompassDirection newData);
23      void cleanExpired();
24      CompassDirectionWithTimeStamp getDataFrom(String id);
25      CompassDirectionWithTimeStamp getDataFromIndex(size_t
            index);
26  };
27
28  #endif
```

```
1   #include "DirectionsManager.h"
2   #include "Logger/OnroLogger.h"
3
4   DirectionsManager::DirectionsManager(int pinMotor1, int
        pinMotor2, float compassTolerance, float
        declinationAngle) {
5       this->disabled = false;
6       this->pinMotor1 = pinMotor1;
7       this->pinMotor2 = pinMotor2;
8
9       this->compass   = new Compass(declinationAngle);
10      this->movementTolerance = compassTolerance;
11      this->currentTolerance = compassTolerance;
12
13      pinMode(this->pinMotor1, OUTPUT);
14      pinMode(this->pinMotor2, OUTPUT);
15
16      updateCurrentAngle();
17  }
18
19  float DirectionsManager::headFoward(float targetAngle) {
20      if(this->disabled){
21          digitalWrite(this->pinMotor1, LOW);
22          digitalWrite(this->pinMotor2, LOW);
23      } else {
```

```
24          updateCurrentAngle();
25          OnroLogger::getInstance().logln("
               DirectionsManager - Going from "+String(this
               ->currentAngle)+" to "+String(targetAngle));
26
27          if(isCloseEnough(targetAngle)){
28              OnroLogger::getInstance().logln("
                   DirectionsManager - Close enough");
29              this->currentTolerance = this->
                   movementTolerance;
30              digitalWrite(this->pinMotor1, HIGH);
31              digitalWrite(this->pinMotor2, HIGH);
32          } else {
33              OnroLogger::getInstance().logln("
                   DirectionsManager - Positioning");
34              this->currentTolerance = this->
                   movementTolerance / 4.0;
35              position(targetAngle);
36          }
37      }
38  }
39
40  float DirectionsManager::position(float targetAngle) {
41      float diff = targetAngle - this->currentAngle;
42      if(diff >= 0) {
43          float dist1 = diff;
44          float dist2 = 360 - diff;
45          rotate(dist1,dist2);
46
47      } else {
48          float dist1 = - diff;
49          float dist2 = 360 + diff;
50          rotate(dist2,dist1);
51      }
52
53      delay(50);
54      digitalWrite(this->pinMotor1, LOW);
55      digitalWrite(this->pinMotor2, LOW);
56      delay(50);
57  }
58
59  void DirectionsManager::updateCurrentAngle(){
60      this->currentAngle = compass->getAngle();
61  }
62
63  float DirectionsManager::getCurrentDir() {
64      this->updateCurrentAngle();
65      return this->currentAngle;
66  }
```

```
67
68  void DirectionsManager::rotate(float dist1, float dist2)
        {
69      if(dist1 >= dist2) {
70          digitalWrite(this->pinMotor1, HIGH);
71          digitalWrite(this->pinMotor2, LOW);
72      } else {
73          digitalWrite(this->pinMotor1, LOW);
74          digitalWrite(this->pinMotor2, HIGH);
75      }
76  }
77
78  bool DirectionsManager::isCloseEnough(float targetAngle)
        {
79      float diff = abs(targetAngle - this->currentAngle);
80
81      if(diff > 270)
82          diff = 360 - diff;
83
84      bool result = diff < this->currentTolerance;
85
86      return result;
87  }
88
89  void DirectionsManager::disable() {
90      this->disabled = true;
91  }
92
93  void DirectionsManager::enable() {
94      this->disabled = false;
95  }
```

```
1   #ifndef DIRECTIONS_MANAGER_H
2   #define DIRECTIONS_MANAGER_H
3
4       #include "application.h"
5       #include "Compass.h"
6
7       class DirectionsManager{
8           public:
9               DirectionsManager(int pinMotor1, int
                    pinMotor2, float compassTolerance, float
                    declinationAngle);
10
11              float headFoward(float angle);
12              void disable();
13              void enable();
14              float getCurrentDir();
```

```
15
16          private:
17              Compass *compass;
18
19              int pinMotor1;
20              int pinMotor2;
21              float movementTolerance;
22              float currentTolerance;
23              float currentAngle;
24              bool disabled;
25
26              bool isCloseEnough(float targetAngle);
27              void updateCurrentAngle();
28              float position(float angle);
29              void rotate(float dist1, float dist2);
30      };
31
32  #endif
```

```
1   #include "AddressManager/AddressManager.h"
2   #include "Logger/OnroLogger.h"
3   #include "KeepAliveSender.h"
4
5   #include "Constants.h"
6   #include "application.h"
7
8   Thread *keepAliveThread;
9
10  /* Thread in charge of sending a keep alive if the
       current
11   * device is the selected leader.
12   */
13  os_thread_return_t keepAlive(void* param){
14
15      BullyStatus *bullyStatus = (BullyStatus *)param;
16
17      String myId = AddressManager::getMac();
18      String currentLeader = bullyStatus->getLeaderId();
19      while (true) {
20          char buffer[idLength];
21
22          if(currentLeader.equals(myId)){
23              UDP sendUdp;
24              sendUdp.begin(groupPort+1);
25              myId.toCharArray(buffer, idLength);
26              IPAddress remoteIP(groupIp);
27              int port = groupPort;
28              sendUdp.beginPacket(remoteIP, port);
```

```
29              //OnroLogger::getInstance().logln("Keep alive
                    sending "+String(buffer));
30              sendUdp.write(reinterpret_cast<const unsigned
                    char*>(buffer), idLength);
31              sendUdp.endPacket();
32          }
33
34          delay(KEEP_ALIVE_TIMEOUT*1000);
35          currentLeader = bullyStatus->getLeaderId();
36          OnroLogger::getInstance().logln("KeepAliveSender"
                );
37      }
38  }
39
40  KeepAliveSender::KeepAliveSender (BullyStatus *status) {
41      this->bullyStatus = status;
42  }
43
44  void KeepAliveSender::start() {
45      keepAliveThread = new Thread("KeepAliveSender",
            keepAlive, this->bullyStatus);
46  }
```

```
1   #ifndef KEEP_ALIVE_BULY_H
2   #define KEEP_ALIVE_BULY_H
3
4   #include "BullyStatus.h"
5
6       class KeepAliveSender {
7
8           private:
9               BullyStatus *bullyStatus;
10
11          public:
12              KeepAliveSender (BullyStatus *status);
13              void start();
14      };
15
16  #endif
```

```
1   #include "AddressManager/AddressManager.h"
2   #include "LeaderLivenessVerifier.h"
3   #include "Logger/OnroLogger.h"
4   #include "application.h"
5   #include "Constants.h"
6
7   Thread *verifier;
```

```
8
9   /* Thread in charge of checking every a given amount of
10   * time the timestamp of the current leader, and if it's
11   * expired, we assume that the leader is down and select
12   * ourself as the leader
13   */
14  os_thread_return_t verify(void* param){
15
16      BullyStatus *bullyStatus = (BullyStatus *)param;
17
18      UDP receiveUdp;
19
20      receiveUdp.begin(groupPort+2);
21      receiveUdp.joinMulticast(groupIp);
22
23      while(true) {
24
25          delay(LIVENESS_TIMEOUT*1000);
26          OnroLogger::getInstance().logln("
                LeaderLivenessVerifier");
27
28          // We should check every a given amount of
                seconds if the leader is still alive (if I am
                 not the leader)
29          String myId = AddressManager::getMac();
30          String currentLeader = bullyStatus->getLeaderId()
                ;
31          if (! currentLeader.equals(myId)) {
32              int timeStamp = bullyStatus->getTimeStamp();
33
34              int currentTime = Time.now();
35
36              int timeDifference = currentTime - timeStamp;
37
38              if (timeDifference > (LIVENESS_TIMEOUT)) {
39                  /* Last leader alive message was more
                        than LIVENESS_TIMEOUT seconds ago, it
                        's probably dead.
40                   * I proclaim myself as the new leader.
41                   */
42                  bullyStatus->setLeaderId(myId);
43
44                  /* Tell everyone that I think that the
                        leader is dead,
45                   * so I am the new leader.
46                   */
47                  UDP sendUdp;
48                  sendUdp.begin(groupPort+3);
49                  char buffer[idLength];
```

```
50                  myId.toCharArray(buffer, idLength);
51                  IPAddress remoteIP(groupIp);
52                  int port = groupPort;
53                  sendUdp.beginPacket(remoteIP, port);
54                  sendUdp.write(reinterpret_cast<const
                        unsigned char*>(buffer), idLength);
55                  sendUdp.endPacket();
56                  // OnroLogger::logln("LeaderVerifier:
                        Sent multicast message:" + String(
                        buffer));
57              }
58          }
59      }
60  }
61
62  LeaderLivenessVerifier::LeaderLivenessVerifier (
        BullyStatus *status) {
63      this->bullyStatus = status;
64  }
65
66  void LeaderLivenessVerifier::start() {
67      verifier = new Thread("LeaderLivenessVerifier",
            verify, this->bullyStatus);
68  }
```

```
1   #ifndef LIVENESS_VERIFIER_BULY_H
2   #define LIVENESS_VERIFIER_BULY_H
3
4   #include "BullyStatus.h"
5
6       class LeaderLivenessVerifier {
7
8           private:
9               BullyStatus *bullyStatus;
10
11          public:
12              LeaderLivenessVerifier (BullyStatus *status);
13              void start();
14      };
15
16  #endif
```

```
1   #include "AddressManager/AddressManager.h"
2   #include "MovementCoordination.h"
3   #include "Logger/OnroLogger.h"
4   #include "Constants.h"
5   #include "application.h"
```

```
6
7   Thread *coordinator;
8
9   /* Sends the target direction to the proper process.
10   * We use multicast messages, because if one of them is
        lost,
11   * the next one will have the target direction updated.
12   */
13  void sendProcessDirection(String ip, int direction) {
14    UDP sendUdp;
15    sendUdp.begin(portMovement + 1);
16
17    char buffer[compassDirectionLength];
18    sniprintf(buffer, compassDirectionLength, "%03d",
          direction);
19
20    byte ipVector[4];
21    AddressManager::toIp(ip, ipVector);
22    IPAddress remoteIP(ipVector);
23
24    sendUdp.beginPacket(remoteIP, portMovement);
25    sendUdp.write(reinterpret_cast<const unsigned char*>(
          buffer), compassDirectionLength);
26    sendUdp.endPacket();
27  }
28
29  /* Gets the robot (index), whose current direction is
        nearest to the
30   * target one. To do so, it iterates through the list of
        robots
31   * checking which of them are free (i.e. they don't have
        an assigned
32   * address yet) and it returns the closest one.
33   */
34  int getClosestRobotIndex(DataStatus dataStatus, bool
       freeRobot[], int nextDirection) {
35    int closestIndex = 0;
36    int closestAngle = 640; // We choose an angle far
          enough
37    for (int i = 0; i < maxCount; i++) {
38      if (freeRobot[i]) {
39        CompassDirection direction = dataStatus.
              getDataFromIndex(i).direction;
40        int difference = (abs(nextDirection - direction.
              compassDirection))%360;
41        difference = difference > 180 ? 360 - difference :
              difference;
42        if (difference < closestAngle) {
43          closestAngle = difference;
```

```
44          closestIndex = i;
45        }
46      }
47    }
48    return closestIndex;
49  }
50
51  /* Check how many process are alive, and send each of
        them
52   * a message with its updated target direction.
53   */
54  void sendDirectionInfo(DataStatus dataStatus, int *
        targetDirection) {
55    bool freeRobot[maxCount] = {};
56    int processCount = 0; // We save the amount of
          available processes
57
58    for (size_t i = 0; i < maxCount; i++) {
59      CompassDirectionWithTimeStamp data = dataStatus.
            getDataFromIndex(i);
60      if (!data.direction.id.equals("")) {
61        OnroLogger::getInstance().logln("
              MovementCoordination - IP: "+ String(data.
              direction.id) + " Dir: "+String(data.direction.
              compassDirection));
62      }
63    }
64
65    for (size_t i = 0; i < maxCount; i++) {
66      CompassDirectionWithTimeStamp data = dataStatus.
            getDataFromIndex(i);
67      if (!data.direction.id.equals("")) {
68        processCount++;
69        freeRobot[i] = true;
70      } else {
71        freeRobot[i] = false;
72      }
73    }
74
75    for (int j=0; j < processCount; j++) {
76      int nextDirection = j * 360/processCount;
77      int robotIndex = getClosestRobotIndex(dataStatus,
            freeRobot, nextDirection);
78      freeRobot[robotIndex] = false;
79
80      String myIp = AddressManager::getIp();
81      if (dataStatus.getDataFromIndex(robotIndex).direction
            .id.equals(myIp)) {
82        SINGLE_THREADED_BLOCK() {
```

```
83          *targetDirection = nextDirection;
84        }
85      } else {
86        sendProcessDirection(dataStatus.getDataFromIndex(
              robotIndex).direction.id, nextDirection);
87      }
88    }
89  }
90
91  /* Thread in charge of the movement coordination of the
        slaves
92   * and send them their new direction via udp messages
93   */
94  os_thread_return_t coordinate(void* param){
95
96    MovementCoordination *movementCoordination = (
        MovementCoordination *)param;
97    BullyStatus *bullyStatus = movementCoordination->
        getBullyStatus();
98    int *currentDirection = movementCoordination->
        getCurrentDirection();
99    int *targetDirection = movementCoordination->
        getTargetDirection();
100
101   // Compass direction algorithm
102   int  dir = 0; // This should be change with the value
        taken from the compass
103   CompassDirection dataInitial = { AddressManager::getIp
        (), dir };
104   DataStatus dataStatus(dataInitial);
105
106   UDP receiveUdp;
107
108   receiveUdp.begin(groupPortCD);
109   receiveUdp.joinMulticast(groupIp);
110
111   while (true) {
112     int updatedDirection = 0;
113     SINGLE_THREADED_BLOCK() {
114       updatedDirection = *currentDirection;
115     }
116     dataStatus.setData({AddressManager::getIp(),
            updatedDirection});
117
118     char data[dataLength];
119
120     while (receiveUdp.parsePacket() > 0) {
121       receiveUdp.read(data, dataLength);
122       String receivedData = String(data).substring(0,
```

```
            dataLength);
123       OnroLogger::getInstance().logln("
            MovementCoordination - Received data: " +
            receivedData);
124       dataStatus.setData(DataCompassDirection::
            toCompassDirection(receivedData));
125     }
126
127     dataStatus.cleanExpired();
128
129     // Check if I am the current leader and if the leader
          algorithm is already stable;
130     String myId = AddressManager::getMac();
131     String currentLeader = bullyStatus->getLeaderId();
132
133     if (currentLeader.equals(myId) && bullyStatus->
          isStable()) {
134       /* I have been the leader for a fair amount of time
            , so we
135        * can consider that the algorithm has stabilized.
             I will send
136        * each process its new direction.
137        */
138       sendDirectionInfo(dataStatus, targetDirection);
139     }
140
141     OnroLogger::getInstance().logln("MovementCoordination
          ");
142     delay(MOVEMENT_TIME*1000);
143   }
144 }
145
146 BullyStatus* MovementCoordination::getBullyStatus() {
147   return this->bullyStatus;
148 }
149
150 int* MovementCoordination::getCurrentDirection() {
151   return this->currentDirection;
152 }
153
154 int* MovementCoordination::getTargetDirection() {
155   return this->targetDirection;
156 }
157
158 MovementCoordination::MovementCoordination (BullyStatus *
     bullyStatus, int *currentDirection, int *
     targetDirection) {
159   this->bullyStatus = bullyStatus;
160   this->currentDirection = currentDirection;
```

```
161   this->targetDirection = targetDirection;
162 }
163
164 void MovementCoordination::start() {
165   coordinator = new Thread("MovementCoordination",
          coordinate, this);
166 }
```

```
1  #ifndef MOVEMENT_COORDINATION_H
2  #define MOVEMENT_COORDINATION_H
3
4  #include "Bully/BullyStatus.h"
5  #include "Compass/DataStatus.h"
6
7    class MovementCoordination {
8      private:
9        BullyStatus *bullyStatus;
10       int *currentDirection;
11       int *targetDirection;
12
13       int getClosestRobotIndex(bool freeRobot[], int
             nextDirection);
14       void sendProcessDirection(int robotIndex, int
             direction);
15
16     public:
17       MovementCoordination (BullyStatus *bullyStatus, int
              *currentDirection, int *targetDirection);
18       void start();
19
20       BullyStatus *getBullyStatus();
21       int *getCurrentDirection();
22       int *getTargetDirection();
23     };
24
25 #endif
```

```
1  #include "AddressManager/AddressManager.h"
2  #include "MovementDirectionReceiver.h"
3  #include "Logger/OnroLogger.h"
4  #include "Constants.h"
5  #include "application.h"
6
7  Thread *movementDirectionReceiverThread;
8
9  os_thread_return_t movementDirectionReceiver(void* param)
       {
```

```
10
11    int *destinationDirection = (int *)param;
12
13    UDP receiveUdp;
14    receiveUdp.begin(portMovement);
15
16    String myIp = AddressManager::getIp();
17
18    while(true) {
19      char data[compassDirectionLength];
20
21      if (receiveUdp.parsePacket() > 0){
22        receiveUdp.read(data, compassDirectionLength);
23        String receivedData = String(data).substring(0,
              compassDirectionLength);
24
25        int receivedValue = receivedData.toInt();
26        OnroLogger::getInstance().logln("
              MovementDirectionReceiver - Received a new
              direction: "+String(receivedValue));
27        SINGLE_THREADED_BLOCK() {
28          (*destinationDirection) = receivedValue;
29        }
30      } else {
31        delay(1000);
32      }
33    }
34  }
35
36  MovementDirectionReceiver::MovementDirectionReceiver (int
        *destinationDirection) {
37    this->destinationDirection = destinationDirection;
38  }
39
40  void MovementDirectionReceiver::start() {
41    movementDirectionReceiverThread = new Thread("
          MovementDirectionReceiver",
          movementDirectionReceiver, this->
          destinationDirection);
42  }
```

```
1   #ifndef MOVEMENT_DIRECTION_RECEIVER_H
2   #define MOVEMENT_DIRECTION_RECEIVER_H
3
4     #include "Compass/DataStatus.h"
5
6     class MovementDirectionReceiver {
7       private:
```

```
8         int *destinationDirection;
9
10      public:
11        MovementDirectionReceiver (int *
              destinationDirection);
12        void start();
13    };
14
15  #endif
```

```
1   #include "OnroLogger.h"
2   #include "Constants.h"
3
4   OnroLogger& OnroLogger::getInstance()
5   {
6       static OnroLogger instance;
7       instance.init();
8       return instance;
9   }
10
11  void OnroLogger::init() {
12    Serial.begin(baudRate);
13    Time.zone(-3);
14  }
15
16  String OnroLogger::timeStamp() {
17    return Time.format(Time.local(), "%H:%M:%S");
18  }
19
20  void OnroLogger::setDeviceName(String name) {
21    this->deviceName = name;
22  }
23
24  String OnroLogger::getCompleteMessage(String message) {
25    return this->deviceName + " - " + OnroLogger::timeStamp
          () + " - " + message;
26  }
27
28  void OnroLogger::sendNetworkLog(String message){
29    UDP sendUdp;
30    sendUdp.begin(logPort+1);
31    char buffer[250];
32
33    message.toCharArray(buffer, message.length());
34
35    IPAddress remoteIP(logIp);
36
37    sendUdp.beginPacket(remoteIP, logPort);
```

```
38    sendUdp.write(reinterpret_cast<const unsigned char*>(
          buffer), message.length());
39    sendUdp.endPacket();
40  }
41
42  void OnroLogger::logln(String message) {
43    String completeMessage = this->getCompleteMessage(
          message);
44    sendNetworkLog(completeMessage+"\n");
45    //WITH_LOCK(Serial) {
46    //  Serial.println(completeMessage);
47    //}
48
49  }
50
51  void OnroLogger::log(String message) {
52    String completeMessage = this->getCompleteMessage(
          message);
53    sendNetworkLog(completeMessage);
54    //WITH_LOCK(Serial) {
55    //  Serial.print(completeMessage);
56    //}
57  }
```

```
1   #ifndef LOGGER_H
2   #define LOGGER_H
3
4     #include "application.h"
5     /*class OnroLogger {
6       private:
7         static String timeStamp();
8       public:
9         static void init();
10        static void logln(String message);
11        static void log(String message);
12    };*/
13
14    class OnroLogger {
15
16      private:
17
18        String deviceName;
19
20        OnroLogger() {} // Constructor? (the {} brackets)
              are needed here.
21
22        OnroLogger(OnroLogger const&); // Don't Implement
23        void operator=(OnroLogger const&); // Don't
```

```
          implement

      void init();
      String timeStamp();
      String getCompleteMessage(String message);

      void sendNetworkLog(String message);

    public:
      static OnroLogger& getInstance();

      void setDeviceName(String name);
      void logln(String message);
      void log(String message);
};

#endif
```

```
#include "ProximitySensor.h"
#include <stdlib.h>
#include <sstream>

ProximitySensor::ProximitySensor(int echoPin, int trigPin
    ){
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);

  this->trigPin = trigPin;
  this->echoPin = echoPin;
}

long ProximitySensor::getDistance(){
  digitalWrite(this->trigPin, LOW);

  delayMicroseconds(2);
  digitalWrite(this->trigPin, HIGH);
  delayMicroseconds(10);

  digitalWrite(this->trigPin, LOW);

  long duration = pulseIn(this->echoPin, HIGH);
  long distance = (duration/2) / 29.1;
  return distance;
}
```

```
#ifndef PROXIMITY_SENSOR_H
#define PROXIMITY_SENSOR_H
```

```
3
4     #include "application.h"
5
6     class ProximitySensor{
7       public:
8         ProximitySensor(int echoPin, int trigPin);
9         long getDistance();
10
11      private:
12        int trigPin;
13        int echoPin;
14    };
15
16  #endif
```

```
1   #include "AddressManager/AddressManager.h"
2   #include "Logger/OnroLogger.h"
3   #include "ReceiverBully.h"
4
5   #include "Constants.h"
6   #include "application.h"
7
8   Thread *receiver;
9
10  /* Thread in charge of receiving the leader id and
11   * decide what to do depending on the received value
12   */
13  os_thread_return_t receiveMessages(void* param){
14
15      BullyStatus *bullyStatus = (BullyStatus *)param;
16
17      UDP receiveUdp;
18
19      receiveUdp.begin(groupPort);
20      receiveUdp.joinMulticast(groupIp);
21
22      while(true) {
23          char data[idLength];
24
25          if (receiveUdp.parsePacket() > 0) {
26
27              receiveUdp.read(data, idLength);
28              String receivedId = String(data);
29
30              //OnroLogger::getInstance().logln("
                    ReceiverBully - Received new multicast
                    message "+receivedId);
31
```

```
32              if (! receivedId.equals(AddressManager::
                    getMac())) {
33                  String currentLeader = bullyStatus->
                        getLeaderId();
34                  int result = currentLeader.compareTo(
                        receivedId);
35
36                  if (result < 0) {
37                      //OnroLogger::getInstance().logln("
                            ReceiverBully - Someone has a
                            wrong leader");
38                      // Someone has a wrong leader
39                      if (currentLeader.equals(
                            AddressManager::getMac())) {
40                          // I'm the leader, so I should
                                tell everyone
41                          UDP sendUdp;
42                          sendUdp.begin(50007);
43                          char buffer[idLength];
44                          AddressManager::getMac().
                                toCharArray(buffer, idLength)
                                ;
45                          //OnroLogger::getInstance().logln
                                ("Someone has a wrong leader
                                - Sending this id: "+String(
                                buffer));
46                          IPAddress remoteIP(groupIp);
47                          int port = groupPort;
48                          sendUdp.beginPacket(remoteIP,
                                port);
49                          sendUdp.write(reinterpret_cast<
                                const unsigned char*>(buffer)
                                , idLength);
50                          sendUdp.endPacket();
51                      }
52                  } else if (result > 0) {
53                      // New Leader
54                      //OnroLogger::getInstance().logln("
                            ReceiverBully - New leader: " +
                            receivedId);
55                      bullyStatus->setLeaderId(receivedId);
56                  } else {
57                      // Leader Alive
58                      //OnroLogger::getInstance().logln("
                            ReceiverBully - Leader alive: " +
                             receivedId);
59                      bullyStatus->setLeaderId(receivedId);
                             // Timestamp is reseted
60                  }
```

```
61              }
62          }
63          else {
64            delay(1 * 1000);
65          }
66          OnroLogger::getInstance().logln("ReceiverBully");
67
68      }
69  }
70
71  ReceiverBully::ReceiverBully (BullyStatus *status) {
72      this->bullyStatus = status;
73  }
74
75  void ReceiverBully::start() {
76      receiver = new Thread("receiverBully",
           receiveMessages, this->bullyStatus);
77  }
```

```
1
2   #ifndef RECEIVER_BULY_H
3   #define RECEIVER_BULY_H
4
5   #include "BullyStatus.h"
6
7       class ReceiverBully {
8
9           private:
10              BullyStatus *bullyStatus;
11              //Thread *receiver;
12
13          public:
14              ReceiverBully (BullyStatus *status);
15              void start();
16      };
17
18  #endif
```