Delft University of Technology

Extra Project (15 ECTS)

Hardware Design and API Development for a Robotic System

November 28, 2016

Contents

Abstract

Networked Swarm Robots is an upcoming field in the vast domain of robotics. They find vast applications in various domains mainly in disaster management. Swarm Robots must have a certain amount of intelligence so that they can be able to localize themselves and also communicate with other robots using the onboard hardware. In this project, a low-cost robotic platform has been developed that is capable of being controlled from a server computer wirelessly and can also communicate wirelessly with other robots through an ad-hoc network. The ad-hoc network design was not a part of the project and is provided externally. The robot is provided with an onboard MCU that takes care of the processing. The robotic hardware also contains hall effect sensor, a magnetometer and an ultra-sonic sensor for localization. An external Wi-Fi module is provided for wireless communication. The robotic platform developed has been used for the course 'Ad-hoc Networks' at TU Delft.

1 Introduction

Robotics is one of the fastest developing fields today in engineering and technology. Robots are used for many applications ranging from military to health care. Swarm robotics is a sub-field of robotics and is finding vast applications in different fields. In this project, a low-cost robotic platform has been developed that can be utilized to realize swarm robotic applications. The aim of the project is to develop a robotic platform that can communicate with other robots and also to the server in an ad-hoc network. The robotic platform must be flexible enough to adapt to different network topologies and at the same time, it must be easy to program. For this purpose, a set of APIs(Application Programming Interface) must be provided that can take care of the onboard sensors, communication between different nodes and also the locomotion of the robots. Section 2 of the report will describe the hardware architecture of the robotic platforms, section 3 will describe the software architecture of the robotic platform, Section 4 describes the implementation. Section 6 describes the conclusion and the future scope of the project.

2 Hardware Architecture

In this section, the hardware architecture of the system will be described. The hardware architecture is an important aspect of the platform in the sense that it must be flexible for modification and also be cost effective and at the same time provide the required functionality for the robot to be operable in a swarm robotic system. Any robotic system must contain these five components namely, the processor which will act as the brain of the robot, the onboard sensors which must provide position information, the communication interface which must be able to facilitate inter-robot communication and the power supply that must be used to power the entire robot.

2.1 Processor

The processor acts as the brain of the robot. The processor must be able to perform all the actions of the robot such as handling the locomotion of the robot, the communications of the robot and reading the onboard sensors of the robot. The processor must be fairly powerful(and also cheap) since it must also be able to perform additional user commands that will be used for robot localization, control and various other swarm robotic applications. In order to satisfy these requirements, the Arduino Nano was chosen. The Arduino Nano is a small, complete, and breadboard-friendly board based on the ATmega328. The Arduino Nano can be powered via the Mini-B USB connection, 6-20V unregulated external power supply, or 5V regulated external power supply. The power source is automatically selected to the highest voltage source. The ATmega328 has 32 KB, (also with 2 KB used for the bootloader). The ATmega328 has 2 KB of SRAM and 1 KB of EEPROM[1]. The ATMEGA328 also has peripherals that enable serial communication(for the communications chip), I2C(for the inertial measurement unit), 3 timers that enable accurate timing for scheduling of tasks and also a PWM module that can enable speed control of motors. The Nano also has 8 analog inputs, each of which provides 10 bits of resolution. The Nano can be bought from Chinese chipmakers at 2USD. An additional feature that is hugely advantages is the fact that the Nano can be programmed via the Arduino interface using the processing language or C. The Arduino software provides many APIs that enables easy programming and at the same time is also flexible enough to add additional features.

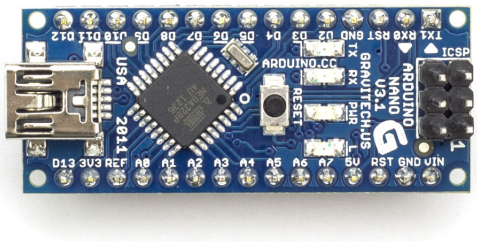


Figure 1: Arduino Nano

2.2 Locomotion and Inertial Measurement Unit

Most of the robots today use differential drive mechanism to provide locomotion.Differential Drive mechanism consists of 2 drive wheels mounted on a common axis, and each wheel can independently being driven either forward or backward. The velocity of each wheel can be varied independently and for the robot to perform the rolling motion, the robot must rotate about a point that lies along their common left and right wheel axis. The point that the robot rotates about is known as the ICC - Instantaneous Center of Curvature[2]. By varying the velocities of the two wheels, we can vary the trajectories that the robot takes. If the velocity of the left wheel is equal to the velocity of the right wheel($V_l = V_r$), then the we have forward motion in a straight line. If $V_l = -V_r$, then we have the rotation about the midpoint, and the robot rotates in place. If$V_l = 0$, then the robot rotates about the left wheel. Same is the case for the right wheel[2].

Table 1: Truth Table for one motor

Input A	Input B	Output A	Output B	Description
LOW	LOW	LOW	LOW	Off
HIGH	LOW	HIGH	LOW	Forward
LOW	HIGH	LOW	HIGH	Reverse

In order to provide locomotion to the robot a four wheeled robot-chassis

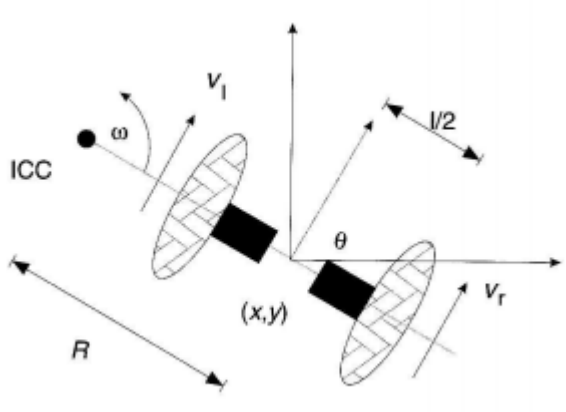


Figure 2: Differential Drive Mechanism [2]

was bought online and a differential drive mechanism was used for the locomotion of the robot. Since the arduino cannot power the motors on the robot, an external motor driver package was connected to the Ardino so that it can drive the motors the way it wants. The HG7881 (L9110) is a compact motor driver chip that supports a voltage range from 2.5-12V at 800mA of continuous current. Each HG7881 (L9110) chip is able to drive a single DC motor using two digital control inputs. One input is used to select the motor direction while the other is used to control the motor speed.

The HG7881 (L9110) Dual Channel Motor Driver Module uses two of these motor driver chips. Each driver chip is intended to drive one motor, so having two means that this module can control two motors independently. Each motor channel uses the same truth table as above(Table 1). In order to rotate the robot using the differential drive mechanism, the motors are powered as shown in Table 2. In addition to the locomotion, the robot must be provided with on-board sensors so that it can localize itself in a location. So, in order to do this, three sensors were chosen to be provided to the robot. They are:

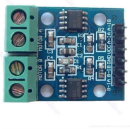
- 3 Ultrasonic Sensors HC-SR04
- 1 Three-Axis, Digital Magnetometer MAG3110
- 2 Hall Effect Sensors

Table 2: Motor Control for robot locomotion

Left Motors	Right Motors	Robot Movement
Forward	Forward	Forward
Forward	Reverse	Rotate Right
Reverse	Forward	Rotate Left
Reverse	Reverse	Reverse

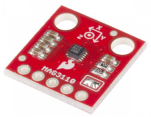
Ultrasonic Sensors were provided so that the robot can detect obstacles. 3 Ultrasonic sensors were provided in 3 directions so that the obstacle can be detected in three directions. The ultrasonic sensors in 3 directions can be used by the robot to localize itself in a hallway or a room. HC-SR04 was chosen since it can be easily integrated with the Arduino.

(a) Robot Chassis with wheels and motors



(b) Motor Driver

(c) HC-SR04 ultrasonic sensor



(d) MAG-3110 sensor

Figure 3: Different Hardware Components of the robot

The HC-SR04 ultrasonic sensor uses sonar to determine the distance of an object like bats. It offers excellent non-contact range detection with high accuracy and stable readings in an easy-to-use package. From 2cm to 400 cm or 1" to 13 feet. Its operation is not affected by sunlight or black material. It comes complete with ultrasonic transmitter and receiver module [3]. It has a resolution of 0.3 cm and a measuring angle of 30 degrees. It operates at 5V which makes it easier to interface with the Arduino. Technical details of HC-SR04 are shown in the Table 3. To start the measurement, the `Trig` pin of SR04 must receive at least 10 μseconds. After this, the sensor will

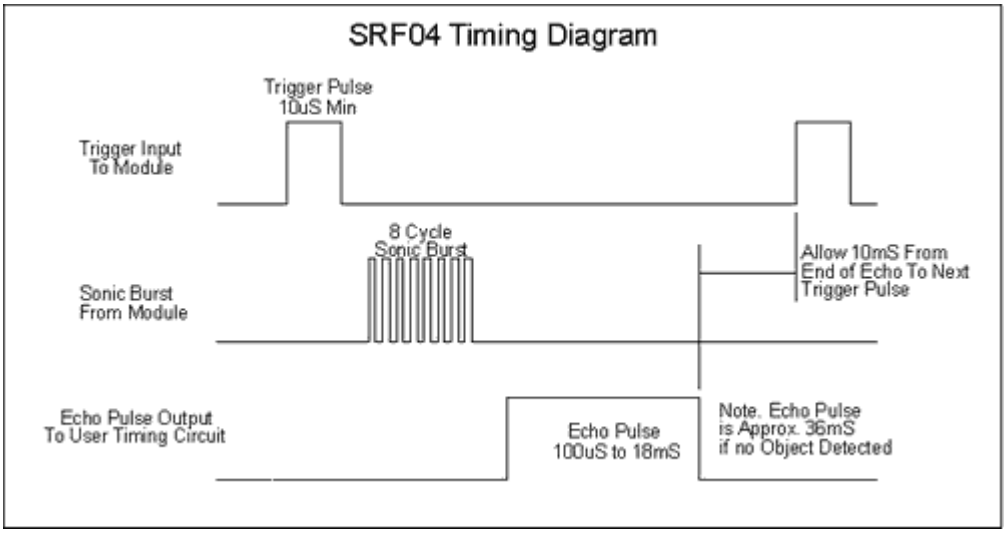


Figure 4: HC-SR04 Timing Diagram

transmit out 8 busts of 40KHz pulse and wait for the reflected ultrasonic burst from the obstacle. When the sensor detected ultrasonic from receiver, it will set the Echo pin to high (5V) and delay for a period (width) which proportion to distance. To obtain the distance, measure the width (Ton) of Echo pin. The following formula was used to calculate the distance of the obstacle from the sensor[3]. The timing diagram of the ultrasonic sensor is shown in Figure 4.

$$Distance(in\ cm) = \frac{Width\ of\ echo\ pulse\ in\ \mu seconds}{58}$$

Table 3: HC-SR04 Technical Specifications

Parameter	Min.	Typ.	Max
Operating Voltage	4.5V	5V	5.5V
Quiescent Current	1.5mA	2mA	2.5mA
Working Current	10mA	15mA	20mA
Ultrasonic Frequency	-	40KHz	-

The next sensor that was provided to the robot in order to localize itself is MAG3110 Three-Axis, Digital Magnetometer. The MAG3110 is capable of measuring magnetic fields with an output data rate (ODR) up to 80 Hz; these output data rates correspond to sample intervals from 12.5 ms to several seconds.The MAG3110 can be used by the robot to calculate it's heading with respect to the magnetic north. This heading angle can be used find the angle rotated by the bot and also be used for localization with in a room. The MAG3110 comes with a I^2C interface. Since the Arduino Nano has I^2C peripheral, interfacing it with the MCU should also be convenient without any hassles. The master (or MCU) transmits a start condition (ST) to the MAG3110, followed by the slave address, with the R/W bit set to "0" for a write, and the MAG3110 sends an acknowledgement. Then the master (or MCU) transmits the address of the register to read and the MAG3110 sends an acknowledgement. The master (or MCU) transmits a repeated start condition (SR), followed by the slave address with the R/W bit set to "1" for a read from the previously selected register. The MAG3110 then acknowledges and transmits the data from the requested register. The master does not acknowledge (NAK) the transmitted data, but transmits a stop condition to end the data transfer[5]. In the Arduino platform, the I^2C communication functions are taken care of by the `Wire` library. The corresponding registers containing the magnetic fields in the three directions are read by the Arduino using the `Wire` library as mentioned above. A compass heading can be determined by using just the Hx and Hy component of the earth's magnetic field, that is, the directions planar with the earth's surface. This is shown in Figure 5. The expected H_x and H_y values for different orientations is shown in Figure 6(a). The real time magnetometer data for a rotation of 90 degrees is shown in Figure 6(b). The magnetic compass heading can be determined (in degrees) from the magnetometer's x and y readings by using the following set of equations[4]:

$$Direction(y > 0) = 90 - arcTan(H_x/H_y) * 180/\pi$$

$$Direction(y < 0) = 270 - arcTan(H_x/H_y) * 180/\pi$$

$$Direction(y = 0, x < 0) = 180$$

$$Direction(y = 0, x > 0) = 0$$

These angles must be subtracted from the declination mangles. Declination angles vary from place to place and the correct declination angle for Delft can

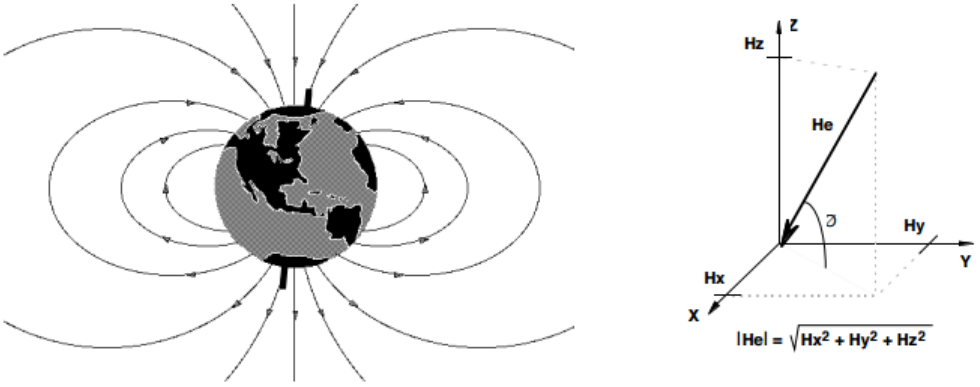


Figure 5: Magnetic Components due to the Earth's magnetic field[5]

be found from [6] as 0.22 degrees. One more way to do this without actually subtracting these values is calibrating the magnetometer properly. In order to calibrate the magnetometer, the maximum and the minimum values of H_x and H_y are found. The bias value is found as:

$$H_{x_{bias}} = \frac{H_{x_{max}} + H_{x_{min}}}{2}$$

$$H_{y_{bias}} = \frac{H_{y_{max}} + H_{y_{min}}}{2}$$

$$H_{x_{scale}} = \frac{1}{H_{x_{max}} - H_{x_{min}}}$$

$$H_{y_{scale}} = \frac{1}{H_{y_{max}} - H_{y_{min}}}$$

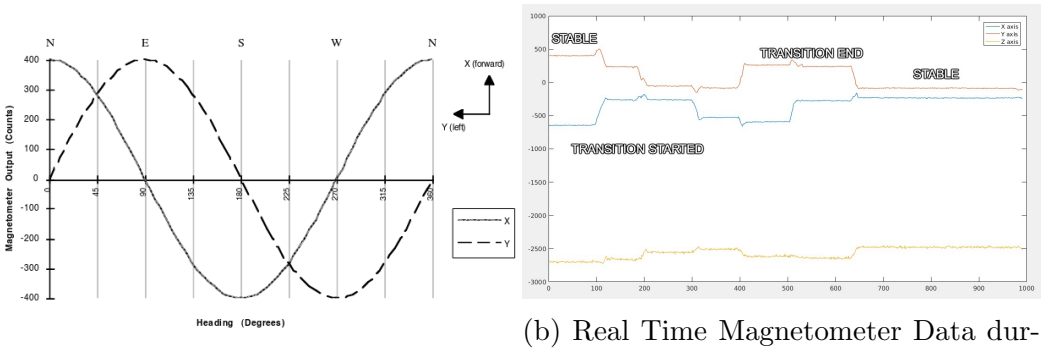
$$H_{x_{calib}} = (H_x * H_{x_{scale}}) - H_{x_{bias}}$$

$$H_{y_{calib}} = (H_y * H_{y_{scale}}) - H_{y_{bias}}$$

The direction can be found as :

$$Direction = arcTan(H_{y_{calib}}/H_{x_{calib}}) * 180/\pi$$

The above equation gives direction in the range of -180 degrees to 180 degrees. So, whenever the direction is less than 0, 360 is added to it, so that the magnetic heading is received in the range of 0 to 360.



(a) H_x,H_y values for different headings (b) Real Time Magnetometer Data during rotation by 90 degrees

Figure 6: H_x and H_y values from magnetometer

The next sensor that was provided to the robot was two hall effect sensors A3144. These sensors were provided so that the robot can calculate the distance it has moved for localization applications. The output of these devices (pin 3) switches low when the magnetic field at the Hall element exceeds the operate point threshold (B_{OP}). At this point, the output voltage is $V_{OUT(SAT)}$. When the magnetic field is reduced to below the release point threshold (B_{RP}), the device output goes high. The difference in the magnetic operate and release points is called the hysteresis (B_{hys}) of the device. This built-in hysteresis allows clean switching of the output even in the presence of external mechanical vibration and electrical noise.Pin 1 of the sensor is connected to +5V and the pin 2 is connected to the ground, Pin 3 is connected to the Vcc through a 10k resistor. This pin is connected to one of the input pins of the Arduino. This is shown in Figure 7. The distance is calculated using the formula:

$$Distance\ moved\ by\ the\ robot(in\ cm) = Falling\ Edge\ count * Diameter\ of\ the\ wheel * \pi$$

Two hall-effect sensors are provided to the robot. Before the distance is measured, both the wheels are aligned using the hall-effect sensors and the measurement is started.

2.3 Communication

The next component that needs to be looked into is the communication between the robots. For this, the ESP8266 was used. The robots communicate

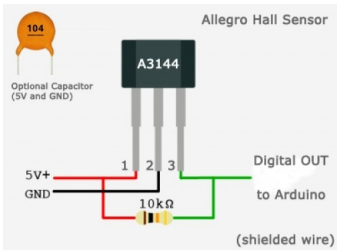


Figure 7: A3144 Circuit Diagram

with the server and other robots through an ad-hoc network. The ad-hoc network design and the ESP8266 firmware design were done externally and was not a part of this project. The packet format that needs to be followed for communication is shown in Figure 8. The packets are created by the Arduino in this particular format and sent to other Arduinos through the ad-hoc network. The bot needs to be able to send packets to the other robots and the server through this format and also be able to parse the packets and handle commands from the other robots or the server using this format. The Arduino is connected to the ESP8266 through a serial UART connection. The description for each of the fields is listed as below:

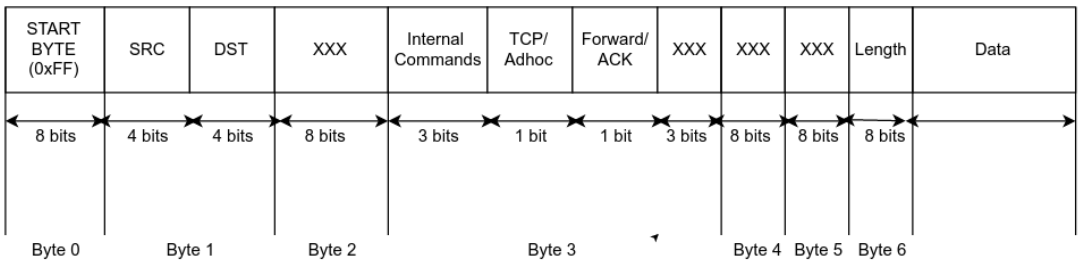


Figure 8: Packet format

- **START BYTE** : It's the first byte. Represents the start sequence of the packet. Its is denoted by 0xFF.
- **SRC** : A 4-bit field of the second byte represents the unique identifier of the source robot.

- **DST** : A 4-bit field of the second byte represents the unique identifier of the destination robot.
- **Internal Commands** : This 3-bit field is used for communication between Arduino and ESP. It represents certain commands that are exchanged between Arduino and ESP during initialization. The internal command should be set to '0' when communicating with TCP or Ad-hoc node (another robot). It should be set only when the robot communicates with its ESP8266 module.
- **TCP/Adhoc** : 1-bit information used to identify whether the packet is intended for TCP server or Ad-hoc network.
  Note: When communicating with TCP, TCP bit should be made high and data should be packed as per the length. Rest of the bytes such as SRC, DST, Internal commands, Forward/ACK are don't cares.
- **Forward/ACK** : 1-bit information used to identify if the packet is ACK packet or a forwarding packet.
- **Length** : Represents the length of data in bytes present in the packet.
- **Data** : Represents data to be sent. This field can be up to 254 bytes in size.
- **Checksum** : Represents checksum to make sure that the complete packet is received. At present, this is not implemented in the server, ESP8266 or Arduino.

The complete hardware architecture of the system is as shown in Figure 9. The cost of the different components of the robot is shown in the Table 4. From the table, it can be seen that the entire cost of the robot comes to about 35.31 USD. The designed robot with its functionalities is much more cheaper than the ones available in the current market. A robot with all these components bought externallu would cost about 100USD. On the whole, the hardware architecture is cost-effective and at the same time also cheap.

3 Software Architecture

The software architecture of the robot consists of three components. They are :

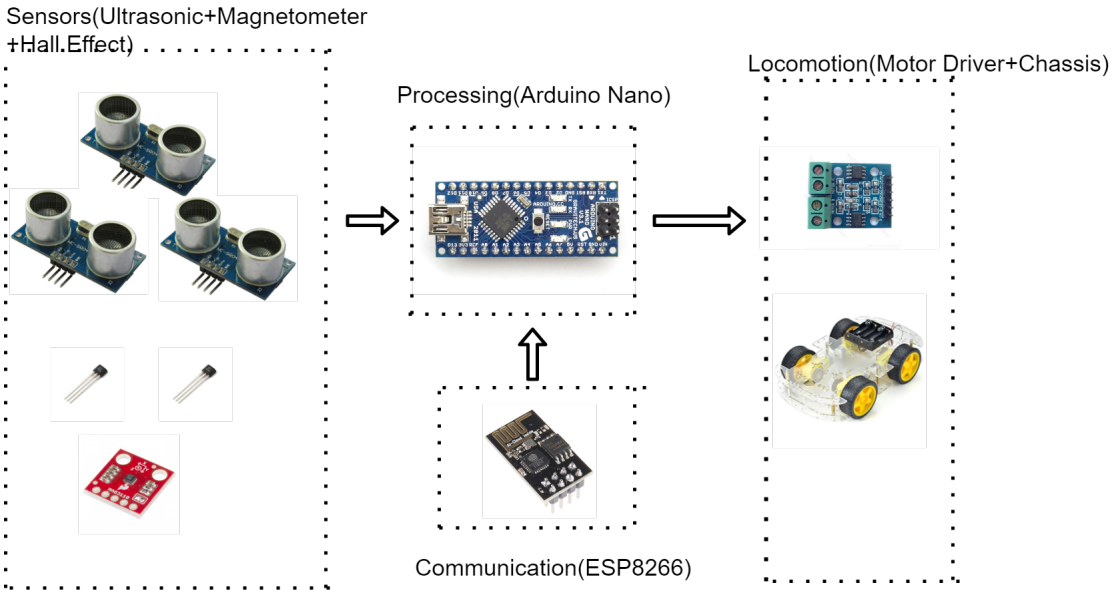


Figure 9: Hardware Architecture of the system

- Application Code : The application code is where the user initializes the GPIO, interrupts and the communication modules using the APIs provided. The user can also use the APIs in the application code to communicate with other bots and also read sensor data.
- Interrupt Handlers: The interrupt handler is called every 50 ms. The interrupt handler executes commands using the information from the communication handler.
- Communication handler: The communication handler parses packets from the other robots and updates the command information so that the interrupt handler executes the command the next time the program runs. The communication handler also sends the packets to the other robots through the interrupt handler. The inter-dependencies between the three components of the code is shown in Figure 11.

As mentioned previously, the interrupt handler is called every 50 ms. It is responsible for the locomotion functions. The communication handler provides the command to be executed. The locomotion functions such as `moveForward`, `moveForwardForTime`, `moveForwardDistance` are handled by

Table 4: Cost of entire Bot

Component	Qty.	Cost/Unit(USD)	Total Cost
Ultrasonic Sensor(HC-SR04)	3	1.80	5.40
Hall Effect Sensor A3144	2	0.10	0.20
Magnetometer MAG3110	1	2.56	2.56
ESP8266	1	3.12	3.12
Chassis	1	15.10	15.10
Motor Driver HG7881	1	2.23	2.23
Arduino Nano	1	1.70	1.70
Battery Bank(2600mAh)	1	5.00	5.00
Total Cost per Bot	-	-	**35.31USD**

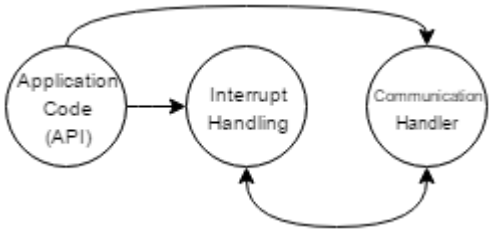


Figure 10: Software Architecture of Bot

this interrupt handler(all directions). Every time a packet is received from another node or the server, the `onPacket()` function is called. This function checks for the start byte 0xFF, updates the RSSI buffer if the ESP sends the INT_RSSI internal command, else it receives the `onReceive()` function which handles the commands. If the command is a a locomotion command, then the `command` variable is set and it is handled the next time the interrupt handler updates. If the command is asking for the sensor data, then the corresponding sensor data is sent back to the server using the `createPacket()` function. The architecture for handling the incoming commands is shown in Figure 11. The application code consists of initializing the GPIOs, Serial communication between the bot and the ESP8266, initializing the timer interrupts and also initializing the magnetometer. After this, a set of internal commands is sent from the Arduino to the ESP. The ID of the robot is sent, the IP address of the server is sent, the communication matrix of the ad-hoc

Figure 11: Software Architecture of Incoming Communication and Packet Handling

network is sent and finally the Wi-Fi SSID and password is sent. The serial communication between the ESP and the arduino is updated using the `PacketSerial.update()` function. The application can also send information to the server or commands to the other robots using the `createPacket()` API function. The software architecture of the application code is shown in Figure 12.

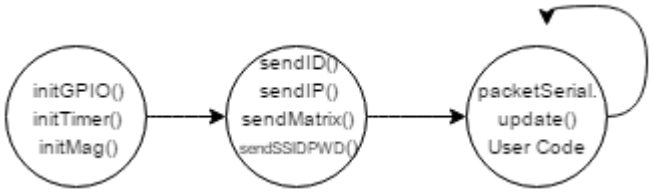


Figure 12: Software Architecture of Main Application

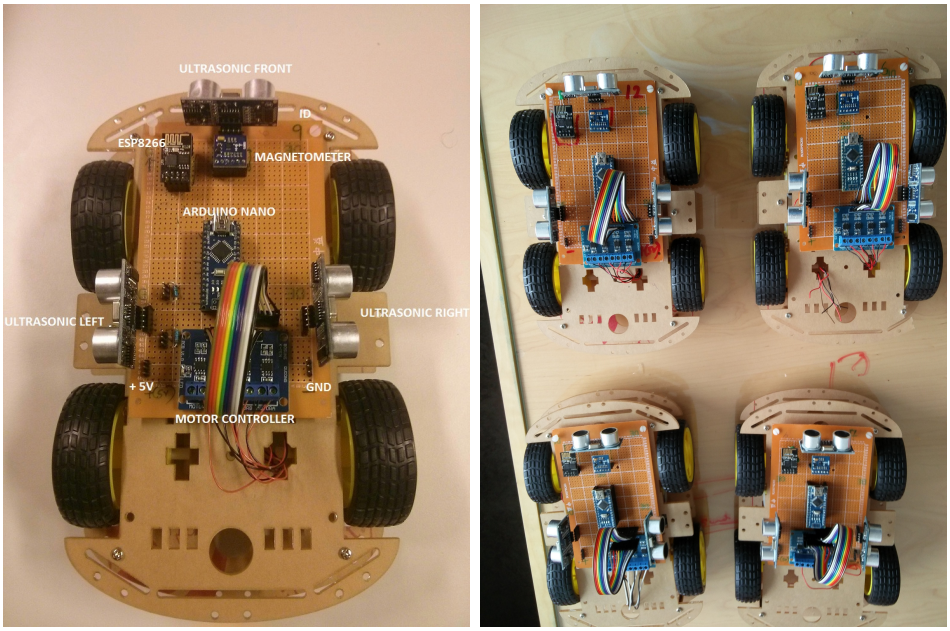
4 Implementation

In this section, the hardware and the software implementation will be described in detail. The connections of the Arduino to the different components in the hardware architecture is tabulated in the Table 5. The motors were connected to the specific pins of D3,D11, D9 and D10 as these pins are connected to the hardware PWM module of the ATMEGA328P. This will enable the users to implement speed control of the robot if they want. The robot currently uses the internal `pulseIn` function of the Arduino calculate the pulse width in-order to do the distance. However, another means of doing so is by using the pin-change interrupts. So, in-order to facilitate this the ultrasonic sensors were connected to the respective pins. The magnetometer is connected to the pins A4 and A5 as these pins are the I2C pins of the ATMEGA328P. The hall-effect sensors are connected to A1 and A2

respectively. The TX pin of the ESP8266 is connected to the RX(D0) of the Arduino and the RX pin of the ESP8266 is connected to the TX(D1) of the Arduino. The connections mentioned above were soldered onto a dotted board and the components were placed on the chassis and attached firmly using a glue. The developed robots and the components are shown in Figure 13a. A group of 10 robots were developed and it was provided to the students of the course 'Ad-hoc networks' at TU Delft. A set of robots are shown in Figure 13b.

Table 5: Connection of Different Components to pins of Arduino

Component	Arduino Pin Connection
Motor Driver Connections	
Left Motor - Positive	D3
Left Motor - Negative	D11
Right Motor - Positive	D9
Right Motor - Negative	D10
Ultrasonic Sensor Connections	
Trigger - U1	D4
Echo - U1	D5
Trigger - U2	D7
Echo - U2	D2
Trigger - U3	D12
Echo - U3	D13
Magnetometer	
SCL	A4
SDA	A5
Hall Effect Sensors	A1,A2
ESP8266	
TX	D0
RX	D1



(a) Different Components of the bot (b) The developed Robots

Figure 13: Developed Robots

The software APIs for the locomotion and the sensors were developed using the techniques mentioned in the previous sections of the report. The set of developed APIs are tabulated in Table 6 and Table 7.

The communication APIs for the communicating with the other robots connected to the Ad-hoc networks is shown in the Table 8. The respective descriptions of the functions are shown in the table.

There are two kinds of commands that will be handled by the robot namely, internal and external. The internal commands represent the communication between the ESP and the Arduino. The internal commands are represented by bit 6 to bit 8 (3 bits) in 5th byte of the packet. These commands are used by Arduino to communicate with ESP8266. The commands are always sent from Arduino to ESP8266. The commands used are:

- INT_ID : Represents that data contains ID
- INT_SSID_PWD : Represents that data contains SSID and Password
- INT_MATRIX : Represents that data contains its connection matrix

Table 6: Locomotion and Sensor APIs

NAME	USAGE	DESCRIPTION
getDistanceFront	unsigned long getDistanceFront()	This API is used to get the distance of the nearest obstacle (in cm) from the front ultrasonic sensor.
getDistanceLeft	unsigned long getDistanceLeft()	This API is used to get the distance of the nearest obstacle (in cm) from the left ultrasonic sensor.
getDistanceRight	unsigned long getDistanceRight()	This API is used to get the distance of the nearest obstacle (in cm) from the right ultrasonic sensor.
moveForward	void moveForward()	This API is used to energize the motors such that the robot moves forward.
stopMotors	void stopMotors()	This API is used to de-energize the motors and stop locomotion of the robot.
moveForwardForTime	void moveForwardFor Time (char *time)	This API is used to make the robot move forward for a specific amount of **time** seconds.
moveBackForTime	void moveBackForTime (char *time)	This API is used to make the robot move back for a specific amount of **time** seconds.
moveBack	void moveBack()	This API is used to energize the motors such that the robot moves back.
turnLeft	void turnLeft (char *time)	This API is used to rotate the robot left for **time** seconds.

Table 7: Locomotion and Sensor APIs

NAME	USAGE	DESCRIPTION
turnRight	void turnRight (char *time)	This API is used to rotate the robot right for **time** seconds.
turnLeftDeg	void turnLeftDeg (char *deg)	This API is used to rotate the robot left by **deg** degrees.
turnRightDeg	void turnRightDeg (char *deg)	This API is used to rotate the robot right by **deg** degrees.
moveForwardDistance	void moveForwardDist (char *dist)	This API is used to energize the motors such that the robot moves forward for **dist**(in cm).
moveBackDistance	void moveBackDist (char *dist)	This API is used to energize the motors such that the robot moves back for **dist**(in cm).
getID	char getID()	This API returns the ID of the robot.
setID	void setID(char ID) (char *time)	This API sets the ID of the robot.
getMagDegrees	int getMagDegrees()	This API returns the magnetic heading of the robot with respect to the magnetic north.

Table 8: Communication APIs

NAME	USAGE	DESCRIPTION
CreatePacket	void sendPacket(char src, char dst,char internal, char isTCP, char isACK, char counterHigh, char counterLow, char dataLength, char *data)	This API can be called to send a packet over WiFi to TCP or ADHOC node. The API forms a packet containing **data** (and also other details such as source ID) and send to TCP/destination (**dst**) robot
OnReceive	void OnReceive(char src, char dst, char internal, char tcp, char fwd, char counterH, char counterL, char datalen, char command, char *data)	This API is a callback function which is called every time a packet is received by Arduino.

- INT_RSSI : Represents that Arduino is requesting RSSI value from ESP8266. The data length can be zero since this is request command
- INT_IP : Represents that data contains IP address of the server

The internal commands are represented in the first byte of data (make sure that Internal commands are set to zero). Followed by external command, arguments are sent in the data section of the packet.

The external commands are sent by the server to a robot in TCP mode, or from one robot to another in ad-hoc mode. The external commands used are:

- NOCOMMAND : Do nothing
- MOVEFORWARD : Command to move forward
- MOVEFORWARDTIME : Command to move forward for specific amount of time
- MOVEBACK : Command to move back
- MOVEBACKTIME : Command to move back for specific amount of time

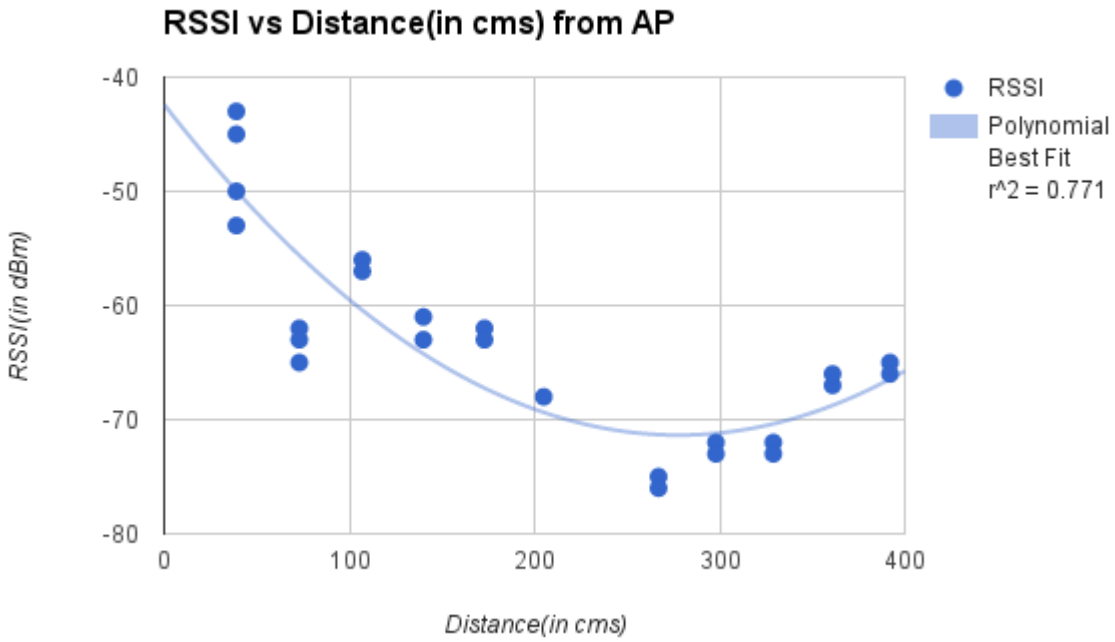


Figure 14: RSSI vs Distance from AP

- TURNLEFT : Command to turn left for specific amount of time
- TURNRIGHT : Command to turn right for specific amount of time
- STOP : Command to stop movement
- DISTANCELEFT : Command to get obstacle distance from left
- DISTANCERIGHT : Command to get obstacle distance from right
- DISTANCEFRONT : Command to get obstacle distance from front
- GETID : Command to get ID of the robot

As an example application, the RSSI was plotted for different distances from the AP by moving the robot. It is shown in Figure 14.

5 Conclusion and Future Work

The developed robotic platform was used in the course 'Ad-hoc Networks' at TU Delft. Some of the bugs in the Arduino code was fixed as the platform was used to develop applications implemented by the students. Some of the experiments conducted were varying the distance of the robots and plotting the RSSI values for different distances. Another experiment involved the robot following an another node(not a robot, but an Arduino Nano with an ESP chip). The third experiment involved the slave car maintaining the same distance from the slave car as the distance of the master car from the AP is varied. More advanced and complex swarm robot applications can be developed using the platform using the APIs and the hardware developed. There is much scope for future work in this project. Localization of different nodes can be performed and an intelligent swarm robotic network application can be developed on top if the platform using this platform and the external ad-hoc network. Personally, this project enabled me to learn various aspects of hardware and software design of networked embedded systems. The robotic platform developed can be considered a starting point for more work in swarm robotics.

References

[1] `https://www.arduino.cc/en/Main/ArduinoBoardNano`

[2] `https://chess.eecs.berkeley.edu/eecs149/documentation/differentialDrive.pdf`

[3] *Product User's Manual – HC-SR04 Ultrasonic Sensor*, v1.0, Cytron Technologies, May 2013

[4] *Compass Heading Using Magnetometers*, Honeywell.

[5] *MAG 3110 Data Sheet*, Freescale Semiconductor

[6] `http://www.ngdc.noaa.gov/geomag-web/`

[7] *A3144 Datasheet*, Allegro MicroSystems.