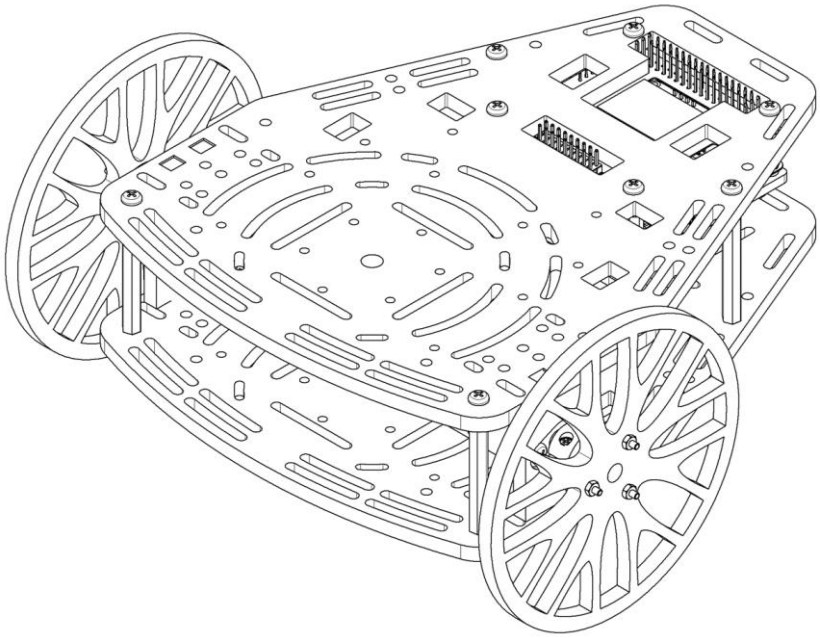
PUZZLE-BOT

BY MANCHESTER ROBOTICS

User's Manual

V 2.1



CONTENTS

WHAT IS PUZZLE-BOT?

WELCOME TO PUZZLE-BOT! BEFORE YOU START MAKING ROBOTICS, YOU'LL NEED TO KNOW SOME BASIC FUNCTIONALITIES OF THE ROBOT.

Puzzle-Bot by Manchester Robotics is a platform for robotics and control systems development and education.

Unlike other products, Puzzle-Bot has sufficient processing capacity to handle multiple advanced add-on components (e.g. different actuators and sensors such as sonar and line following sensors). This allows customers to use a consistent platform for their entire robotics learning journey - from starter kit tutorials all the way to professional prototyping.

The "brain" of Puzzle-Bot is the Development Board. The board acts as a robust computational node for Starter Kit components (e.g. motors) and a communication interface to external computing units (laptops, phones, Raspberry Pi, NVIDIA Jetson Nano, etc.).

It is designed using off-the-shelf components with the aim of achieving the most accessible price point at high-volume production. The Puzzle-Bot Starter Kit contains all the essential components needed to quickly access meaningful robotics capabilities (i.e. a programmable, roaming robot) and provides a user-friendly platform for incorporating a wide range of advanced add-on feature-sets.

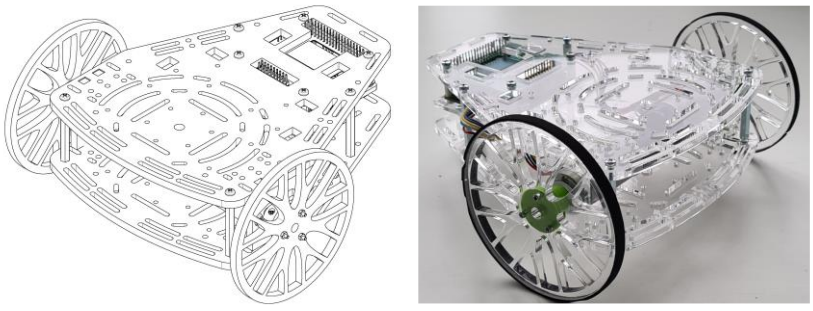


FIGURE 1PUZZLE-BOT BASIC CONFIGURATION

GENERAL SPECS (BASIC VERSION)

Puzzle-Bot comes in two versions, with the same expansion capabilities but different motor types.

- The first version comprises of Brushed DC motors alongside a motor driver for motor control.
- The second version contains brushless motors with a built-in motor driver **(Deprecated Version).**

Brushed DC Motor Version

- Puzzle-Bot control module:
  - ESP32-WROOM-32D microcontroller
    - Xtensa dual-core 32-bit LX6 microprocessor.
    - Memory: 520 KB SRAM
    - 12-bit successive approximation ADC | SAR ADC up to 18 channels
    - 2 × 8-bit digital-to-analog converter | DACs
    - Wi-Fi: IEEE 802.11 | 802.11 b/g/n
    - Bluetooth: v4.2 BR/EDR and BLE
  - DC-DC converter 12 V, 1A.
  - TB6612FNG Dual Motor Driver.
  - 0.96" Inch I2C LCD Module 128X64
- Brushed DC Motor 6V, 170 rpm, 34:1 gear ratio.
- 48 Counts per revolution encoder.
- 5 V, 2.1 A Li-Ion power bank.
- Chassis parts (incl. Screws, standoffs and nuts).
- 3D printed caster wheel holder and steel caster ball.

Brushless Motor Version

- Puzzle-Bot control module:
  - ESP32-WROOM-32D microcontroller
    - - Xtensa dual-core 32-bit LX6 microprocessor.
    - Memory: 520 KB SRAM
    - 12-bit successive approximation ADC | SAR ADC up to 18 channels
    - 2 × 8-bit digital-to-analog converter | DACs
    - Wi-Fi: IEEE 802.11 | 802.11 b/g/n
    - Bluetooth: v4.2 BR/EDR and BLE
  - DC-DC converter 12 V, 1A.
  - 0.96" Inch I2C LCD Module 128X64
- Brushless DC Motor, 12 V, 159 RPM, 45:1 gear ratio.
- 270 Pulses per revolution encoder.
- 5 V, 2.1 A Li-Ion power bank.
- Chassis parts (incl. Screws, standoffs and nuts).
- 3D printed caster wheel holder and steel caster ball.

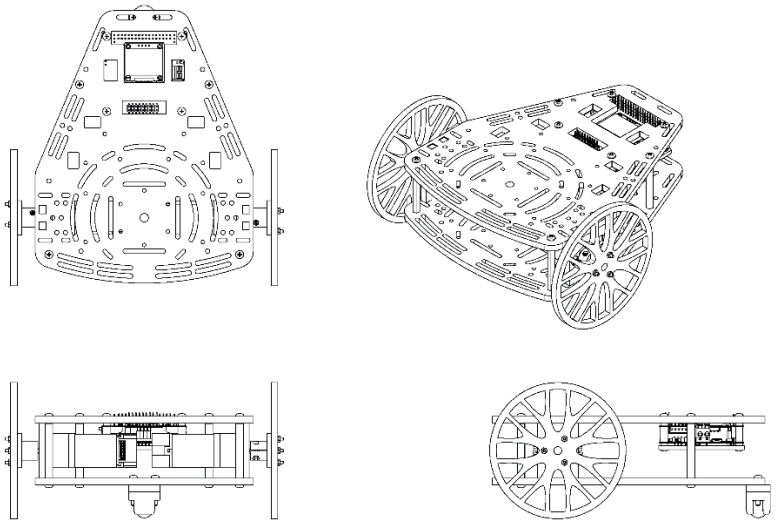


FIGURE 2 PUZZLE BOT GENERAL VIEWS

***For more information regarding the microcontroller, other actuators, sensors, and computational units refer to the appendix.*

BLOCK DIAGRAM

The Puzzle-Bot block diagram is composed as follows

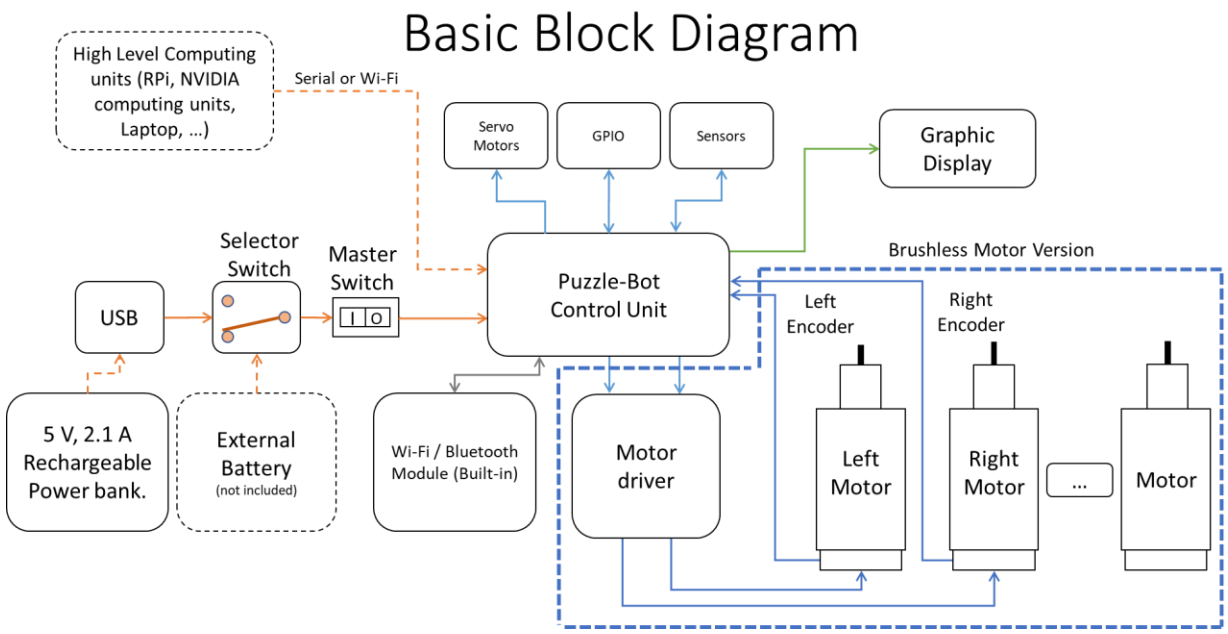


FIGURE 3 BASIC BLOCK DIAGRAM

- The Basic Configuration of the robot comprises of two brushless (deprecated version) or brushed motors (depending on the version), with two encoders (one per motor), a graphic display to show basic information regarding the status of the robot, a Li-ion rechargeable power bank (protected), and pins to connect other sensors or actuators.
- Each motor has a driver embedded for controlling the speed using PWM signals.
- The microcontroller contains built in WIFI and Bluetooth communication.
- External computing units such as (Raspberry pi, Nvidia Jetson, CPU) can be connected to the robot using WIFI or Serial communication.
- Extra sensors and actuators can be connected into the free pins of the microcontroller (servomotors, IR reflectance sensors, sonars etc.).

How does it work?

Puzzle-Bot has two main configuration options. Depending on how it is used, the user can select between the following configurations.

On-Board Configuration

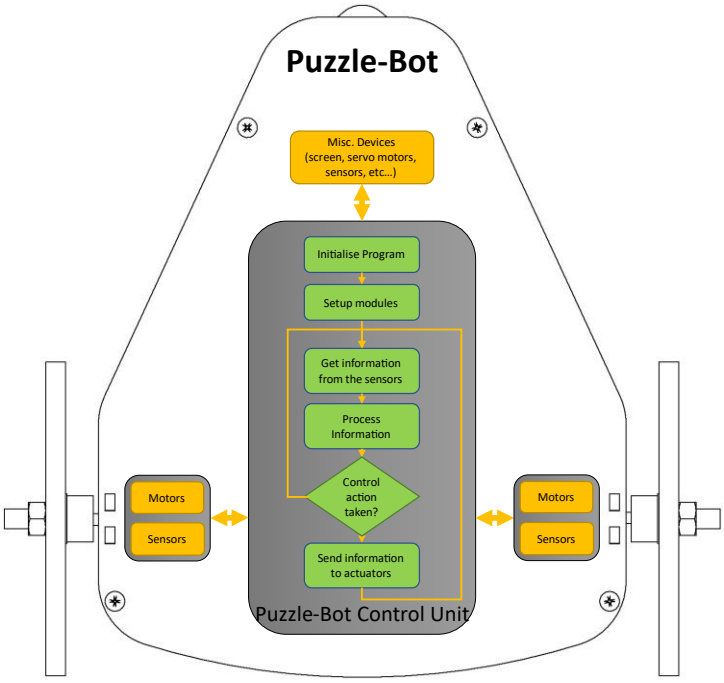


FIGURE 4PUZZLE-BOT ON BOARD CONFIGURATION CONTROL DIAGRAM

- The user uploads their programs directly to the microcontroller inside the Puzzle-Bot.
- Libraries for communication with the different sensors and actuators are provided by Manchester Robotics ltd.
- More peripherals can be attached (e.g., screen, sonar, IR sensor etc.)
- Some basic control libraries are provided.
- More complex controllers can be developed.

External Computing Configuration

- The robot is controlled form an external computing unit.
- The communication is done via WIFI, the connection is done as a normal hotspot.
- The internal firmware and libraries for communicating with the robot are provided by Manchester Robotics.
- The firmware contains libraries for the basic configuration (motors, servomotor, encoders, ultrasonic sensor, IR Reflectance sensor, screen, WIFI communication, etc.)
- The user has access to a basic web interface for testing by using any browser.
- MATLAB libraries for communicating with the sensors and actuators are provided by Manchester Robotics ltd. MATLAB contains an interface for simulating the robot and its capabilities. MATLAB interface does not require any extra libraries for working.
- LabVIEW libraries for communicating with the sensors and actuators are provided by Manchester Robotics ltd. LabVIEW interface also contains a simulator with the basic capabilities of the robot. LabVIEW interface does not require any additional libraries or packages from a third party or National Instruments to work.

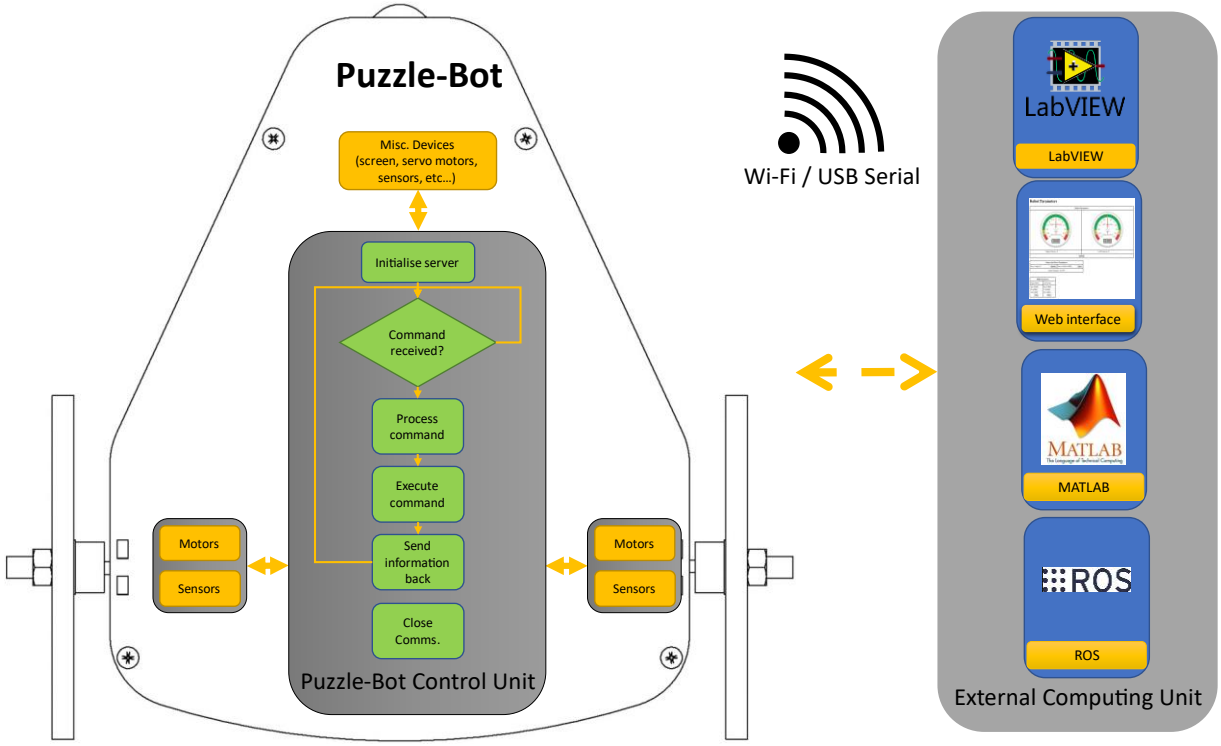


FIGURE 5 PUZZLE-BOT EXTERNAL COMPUTING CONFIGURATION CONTROL DIAGRAM

HARDWARE

BASIC HARDWARE CONFIGURATION

The basic hardware for configuration is divided in two versions depending on the motors used by the Puzzle-Bot. The mechanical and Control Unit diagrams for both versions (brushed DC motors and Brushless motors) are shown in the next sections.

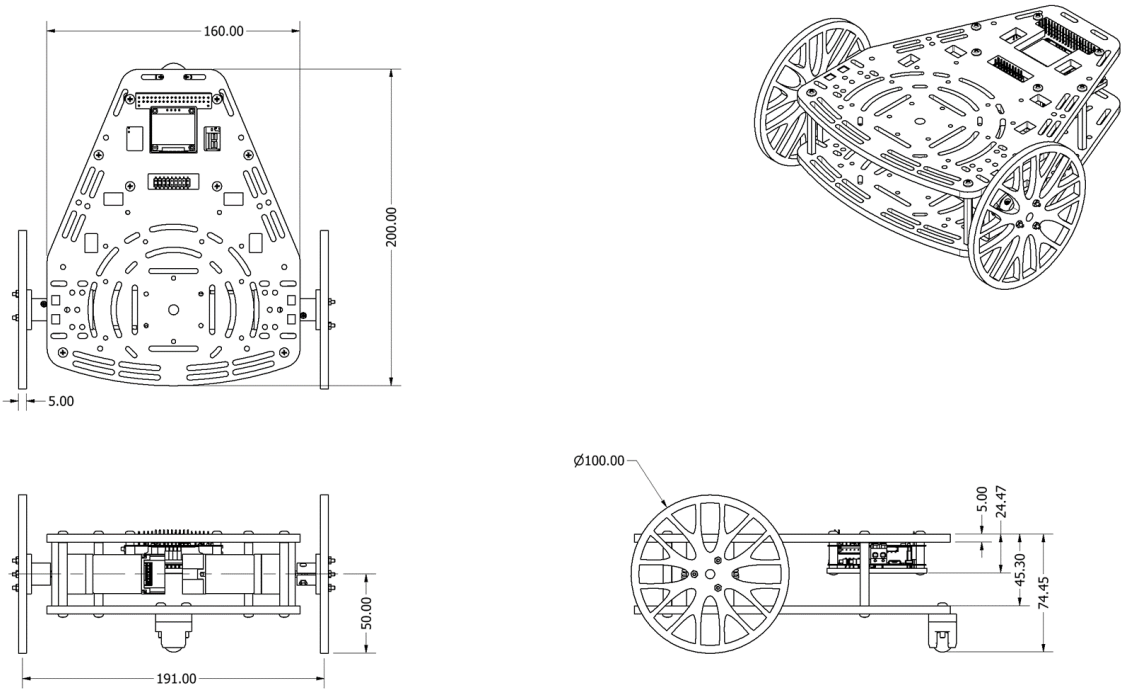


FIGURE 6PUZZLE-BOT GENERAL DIMENSIONS

BRUSHLESS MOTOR VERSION (DEPRECATED VERSION)

The brushless motor version, as described before, consists of two brushless motors, one for each wheel, one Puzzle-bot Control Module (Bottom PCB), one 5V Power Bank, one Caster ball and all the required mechanical components (plastic bases, standoffs, screws and nuts) required for its assembly. The following Diagram shows the basic configuration for the Brushless Motor Version.

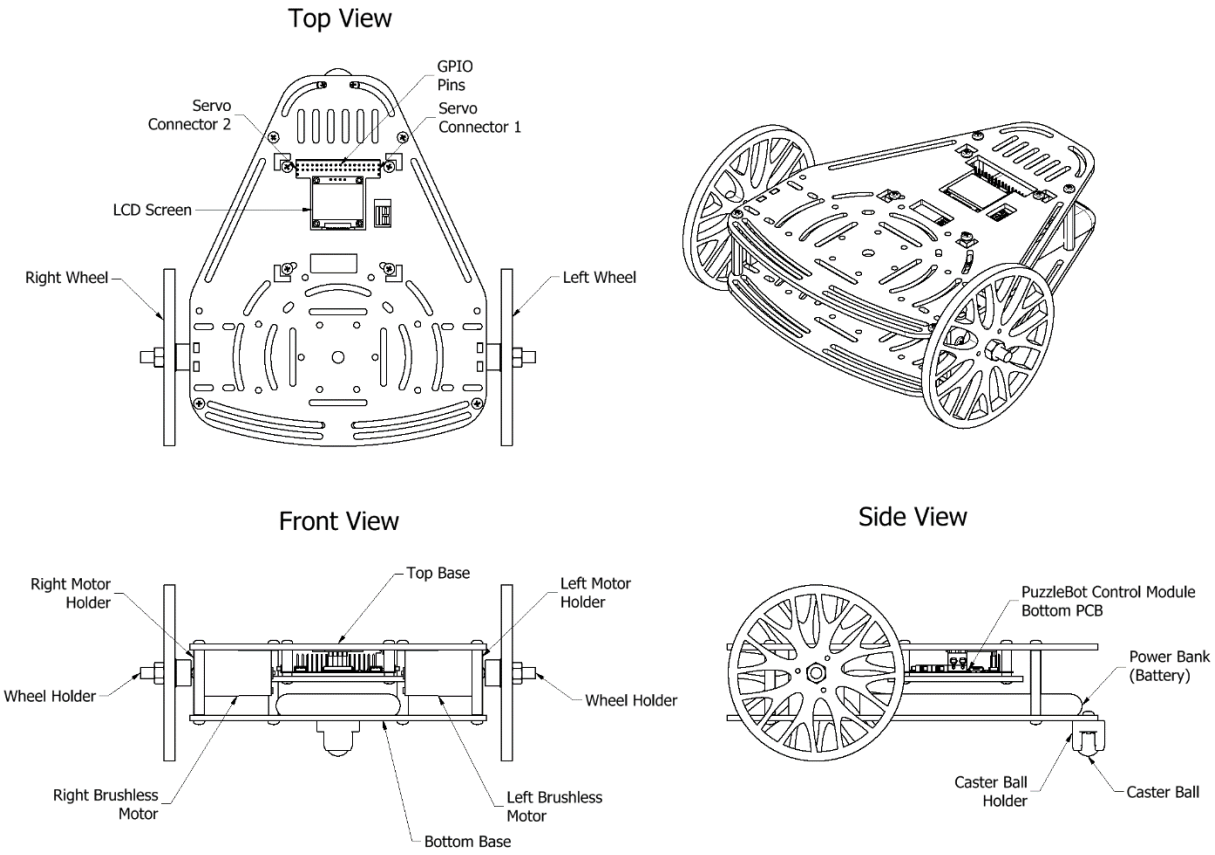


FIGURE 7 PUZZLE-BOT BRUSHLESS MOTOR VERSION

The Puzzle-Bot Control Unit in Figure 10, is based, as described before, on a ESP microcontroller from Espressif Systems. The control Unit is capable of driving three brushless motors, as well as providing different rated voltages 5V and 12 V respectively; GPIO are available for any user intended purposes, and it can be attached to an external (5 – 12 V) power supply.

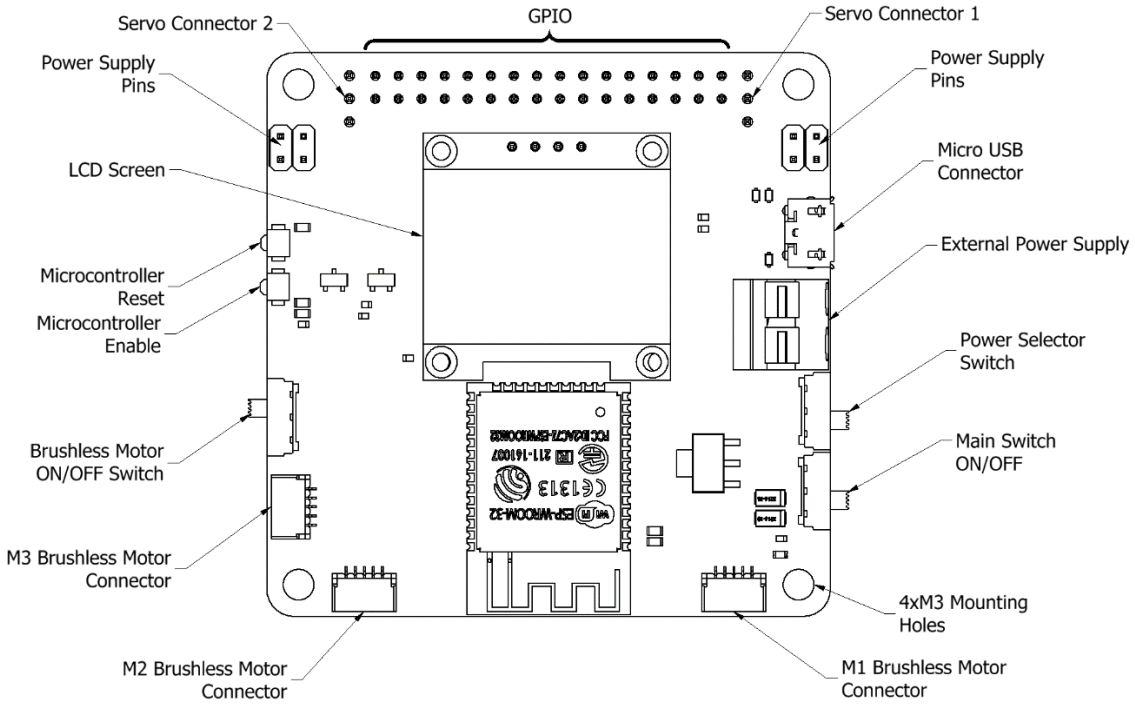


FIGURE 8 PUZZLE-BOT CONTROL MODULE DIAGRAM

TABLE 1 PUZZLE-BOT CONTROL UNIT DESCRIPTION

Main Switch ON/OFF	Main Switch of the Puzzle-Bot
LCD Screen	LCD screen for showing basic information about Puzzle-Bot, or user defined information.
Servo Connector 1,2	Special connectors for servo motors or sensors (user defined), refer to Pinout diagrams section.
GPIO	General Purpose Input Outputs, refer to pinout diagrams section.
Microcontroller Reset	Microcontroller reset button.
Microcontroller Enable	Microcontroller enable button.
Brushless Motor ON/OFF Switch	Security ON/OFF switch for the brushless motors.
M1, M2, M3 Brushless Motor Connector	Brushless motor connectors refer to pinout diagrams section.
Power Selector Switch	Power selector switch (External power supply / USB power supply)
External Power Supply	External power supply input connector.
Micro USB Connector	Programming/Power supply Micro-USB connector.
Power Supply Pins	Power Supply header pins, refer to pinout diagram section.

BRUSHED MOTORS VERSION

The brushed DC motor version, as described before, consists of two DC brushed motors, one for each wheel, one Puzzle-Bot Control Module (Bottom PCB), one Puzzle-Bot DC Motor Control Module, one 5V Power Bank, one caster ball and all the required mechanical components (plastic bases, standoffs, screws and nuts) required for its assembly. The following Diagram shows the basic configuration for the Brushed DC Motor Version.

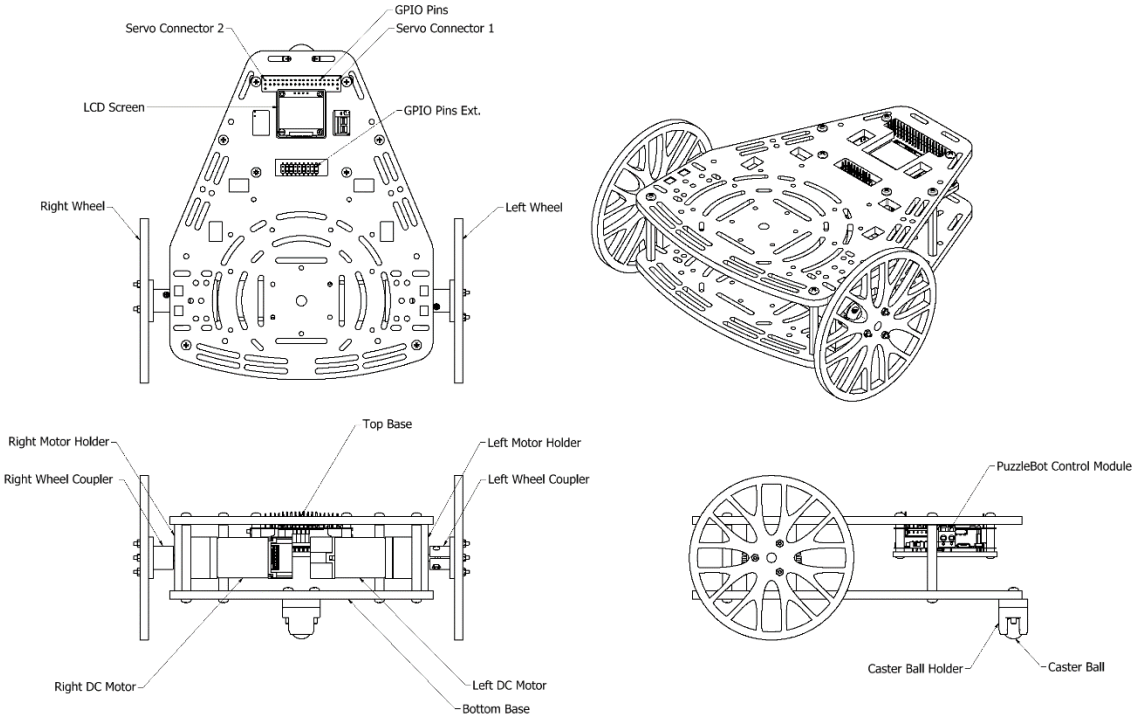


FIGURE 9 PUZZLE-BOT DC BRUSHED MOTOR VERSION

The Puzzle-Bot Control Unit as seen below, is based, as described before, on a ESP microcontroller from Espressif Systems. The control Unit can drive three brushless motors, as well as providing different rated voltages 5V and 12 V respectively; GPIO are available for any user intended purposes, and it can be attached to an external (5 – 12 V) power supply.

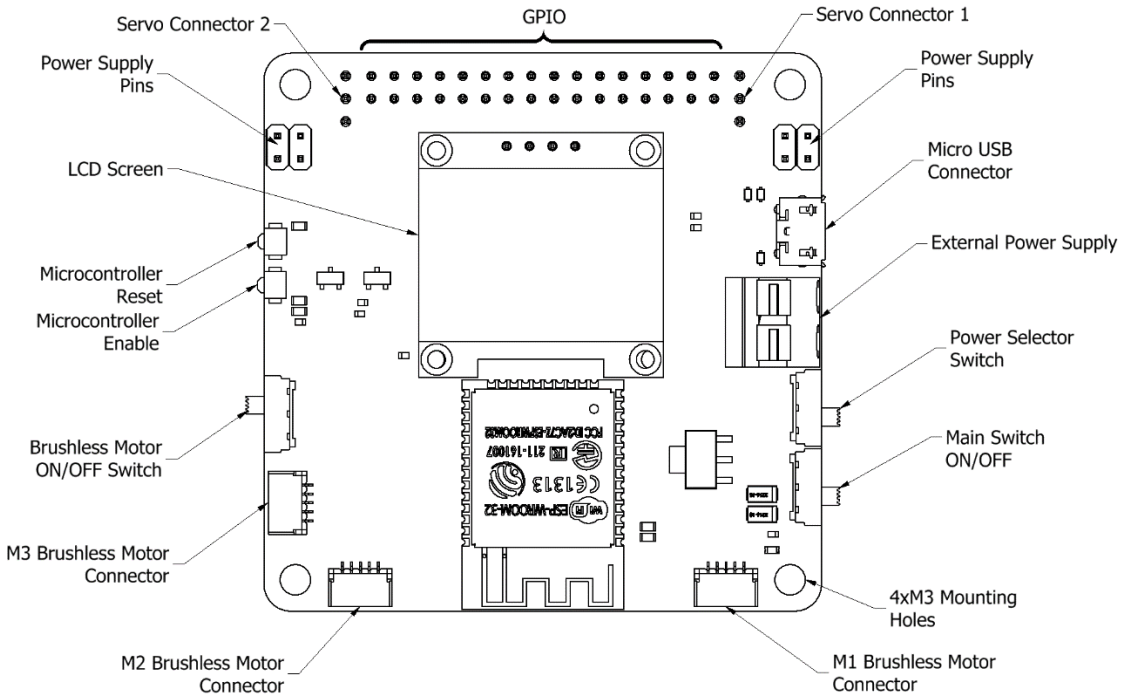


FIGURE 10 PUZZLE-BOT CONTROL MODULE DIAGRAM

**For description about each component refer to Table 1 Puzzle-Bot Control Unit Description.

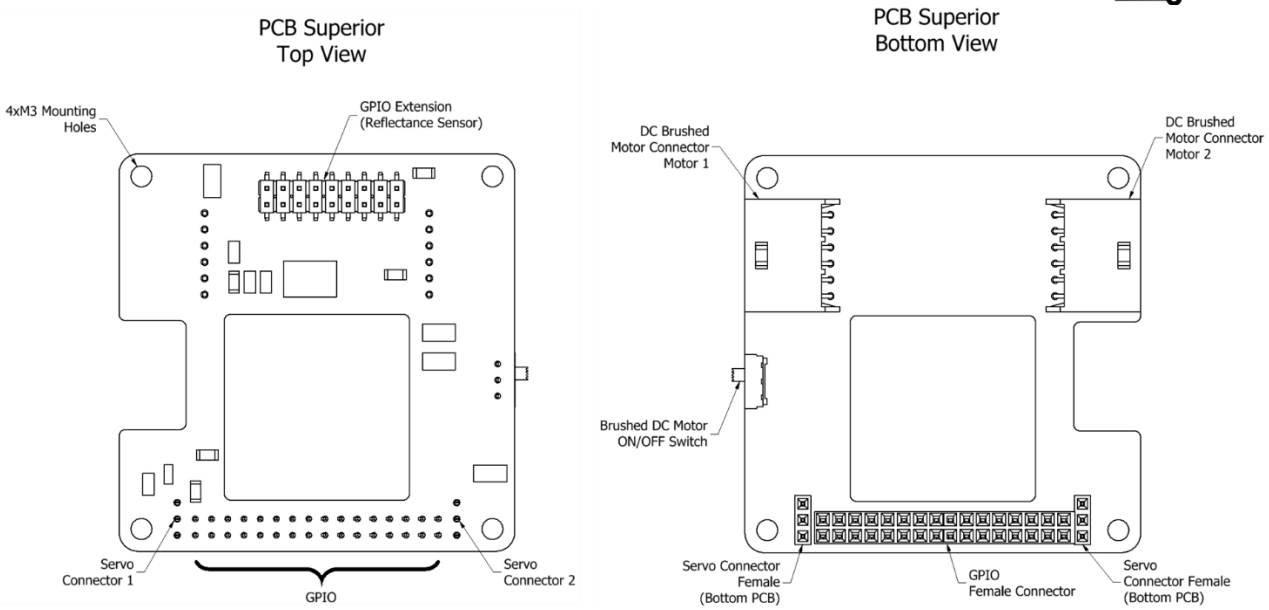


FIGURE 11 PUZZLE-BOT CONTROL MODULE, DC BRUSHED MOTOR DRIVER EXTENSION DIAGRAM (TOP PCB)

TABLE 2 CONTROL MODULE, DC BRUSHED MOTOR DRIVER EXTENSION GENERAL DESCRIPTION

Servo Connector 1,2	Special connectors for servo motors or sensors (user defined), refer to Pinout diagrams section.
GPIO Extension (Reflectance Sensor)	General Purpose Input Output dedicated for easy access when using reflectance sensor (if included). Refer to Pinout diagrams section.
GPIO	General Purpose Input Output, refer to pinout diagrams section.
DC Brushed Motor Connector (Motor 1, Motor 2)	DC Brushed motor connectors refer to pinout diagrams section
Brushed DC Motor ON/OFF Switch	Security ON/OFF switch for the DC brushed motors.

GETTING STARTED (ON BOARD CONFIGURATION)

As described before, the user can directly upload the program to the ESP32 microcontroller. Some basic libraries for communicating with the different sensors and actuators incorporated in the Puzzle-Bot are provided by Manchester Robotics ltd. as well as the instruction to compile such libraries.

IDE SETUP

Although there exists a specific IDE software provided by Espressif Systems which can be used for programming the ESP32, Manchester Robotics ltd. recommends the usage of the Arduino IDE programming environment, for its easy to use user interface and the big community supporting it.

ARDUINO IDE INSTALLATION

In this section a basic tutorial on how to install Arduino IDE will be shown. These instructions are based in *https://www.arduino.cc/en/Guide*

1. Download the Arduino IDE software from *https://www.arduino.cc/*

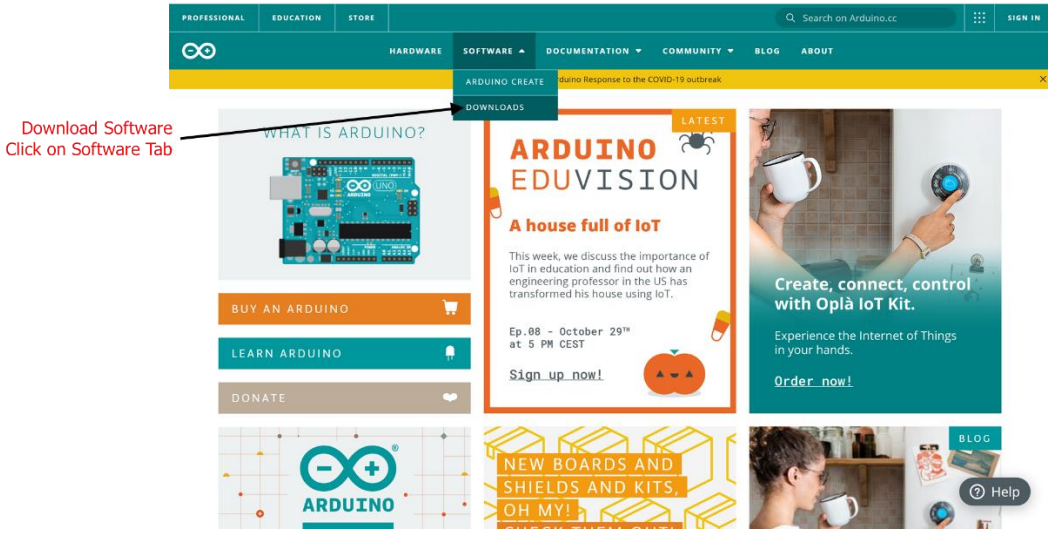


FIGURE 12 ARDUINO IDE DOWNLOAD. **PLEASE NOTE THAT THE WEBPAGE CAN CHANGE

2. Select the operating system where to install the Arduino IDE (Manchester Robotics ltd. does not recommend using the Online IDE)

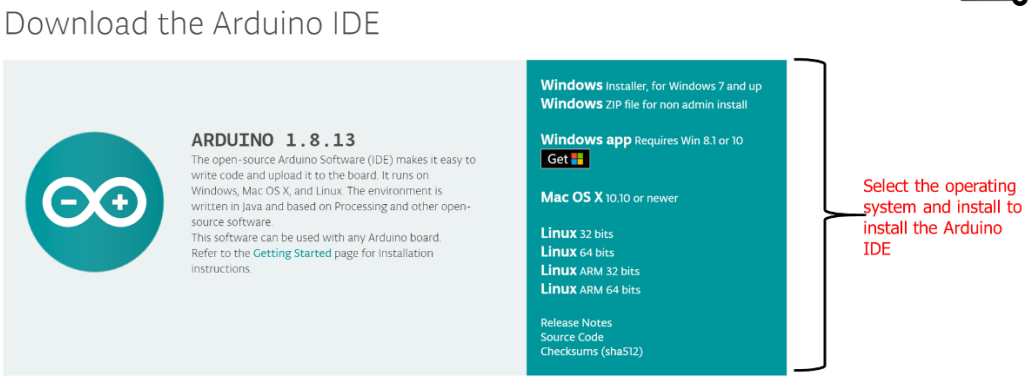


FIGURE 13 SELECT OPERATING SYSTEM AND START DOWNLOAD. **NOTE THAT THIS WEBPAGE CAN CHANGE

3. For each operating system, there is a different set of instructions that can be found in *https://www.arduino.cc/en/Guide* follow the instructions for the selected operating system accordingly.

FIGURE 14 DIFFERENT OPERATING SYSTEM INSTALLATION INSTRUCTIONS. **PLEASE NOTE THE WEBPAGE CAN CHANGE.

4. After the installation is finished, open the program if the installation was successful and the following screen should appear. Manchester Robotics ltd. recommends making some example tutorials and get use to the basics of the Arduino IDE by following the tutorials in *https://www.arduino.cc/en/Guide*.

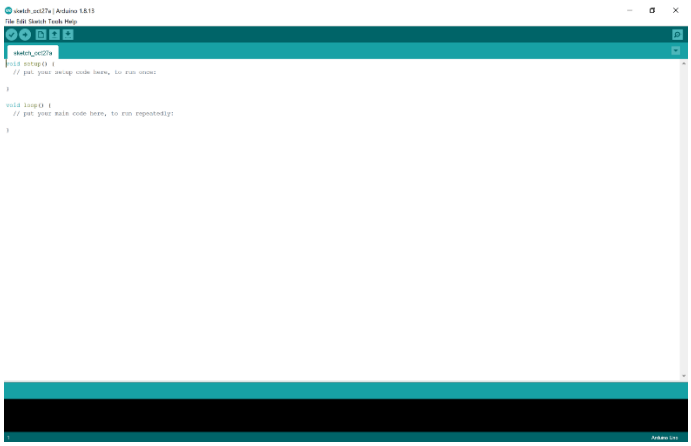


FIGURE 15 ARDUINO IDE, MAIN SCREEN

ARDUINO IDE SETUP

To use the Arduino IDE to program the ESP32, the IDE must be set up before trying to upload any sketch into the microcontroller. The following set up is based on the Web tutorials found in

https://github.com/espressif/arduino-esp32

https://docs.espressif.com/projects/arduino-esp32/en/latest/installing.html

1. In your Arduino IDE, go to File> Preferences

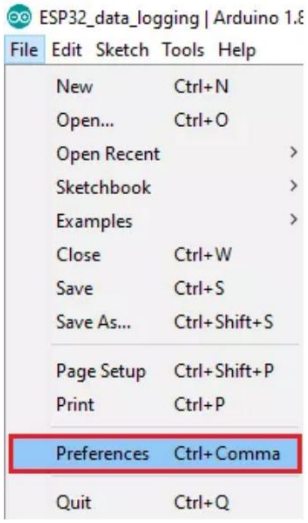


FIGURE 16 ARDUINO IDE PREFERENCES

2. Enter https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json into the "Additional Board Manager URLs" field as shown in the figure below. Then, click the "OK" button.

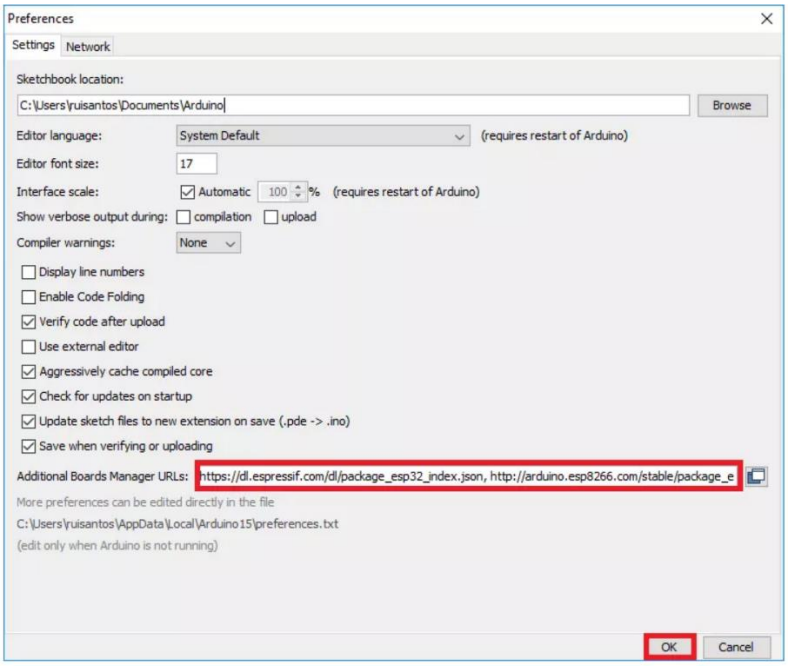


FIGURE 17 ARDUINO IDE PREFERENCES

3. Open the Boards Manager. Go to Tools > Board > Boards Manager…

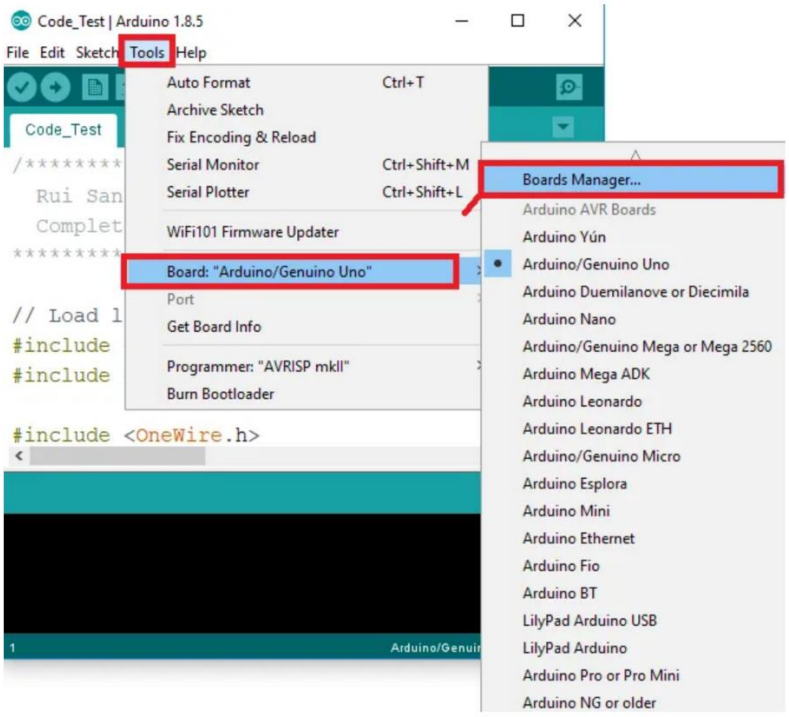


FIGURE 18 ARDUINO BOARD MANAGER

4. Search for ESP32 and press install button for the "ESP32 by Espressif Systems".

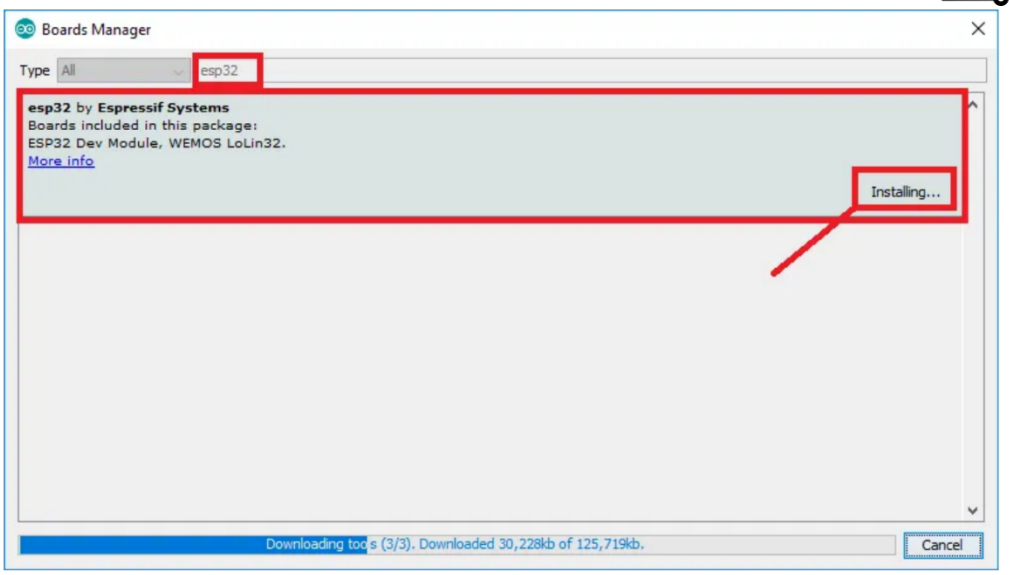


FIGURE 19 ESP32 LIBRARY

5. That's it. It should be installed after a few seconds.

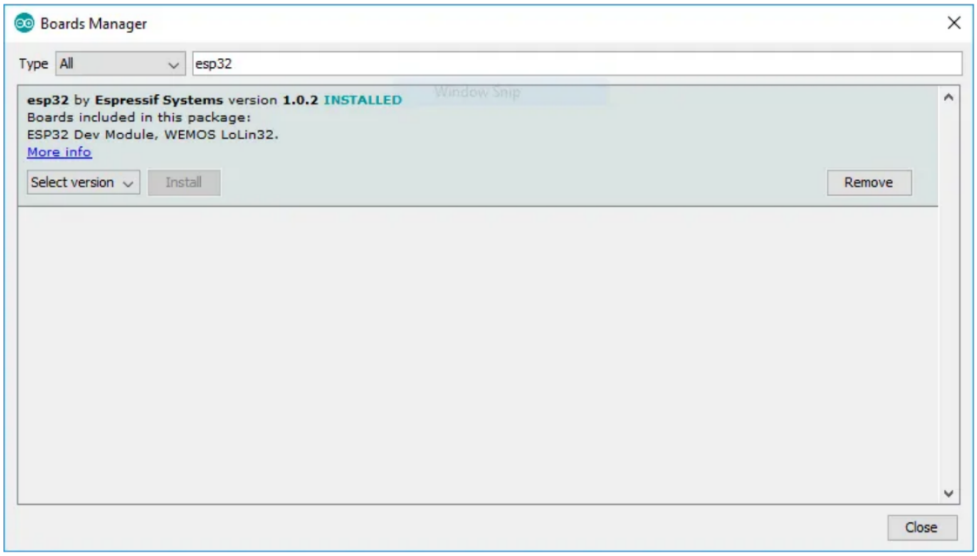


FIGURE 20 ESP32 LIBRARY INSTALLED

6. For testing the installation follow the steps in *https://randomnerdtutorials.com/installing-the-esp32-board-in-arduino-ide-windows-instructions/*

CREATING A BASIC PROGRAM

In this section a basic program example for reading information from the motor encoders and sending different speeds into the motors will be shown using the motor and encoder libraries provided by Manchester Robotics ltd. As stated previously, Manchester Robotics ltd provides some basic libraries for controlling the different actuators and sensors included with the Puzzle-Bot, however the user can build their own libraries if necessary. For further information regarding the libraries visit the Manchester Robotics ltd. Git repository.

Note for further details on how to add libraries and compiling the software or further Arduino programming references, please refer to the Arduino Tutorials webpage https://www.arduino.cc/en/Tutorial/HomePage.

Note: For datasheets and connections corresponding to the Motors, Encoders, Servo Motor, Sonar and Reflectance sensor, check the Appendix.

Note: The libraries and programs used for the following examples can be found in the folder On Board Configuration Examples.

DC BRUSHED MOTOR VERSION

```
/**
   \file Example.ino
   \author Eduard Codres, Mario Martinez
   \copyright Manchester Robotics
   \date Octiber, 2020
   \brief Example program for reaiding and writing motor velocities.
*/

//Inlcude the libraries and header files to be used, for this case
#include "Encoder.h"
#include "MotorDriver.h"


using namespace std;

//Define the global variables to configure the motors and encoders

//Right Motor Configuration Variables
int motorR_type = 2;                //Type of motor 1-brushless, 2 DC Brushed
int motR_pins[3] = {4, 15, 18};     //Define the Motor Pins
int motR_sign = -1;                 //Define the motor rotation sign
float gear_ratio_r = 34;            //Define gear ratio

//Left Motor configuration variables
int motorL_type = 2;
int motL_pins[3] = {2, 12, 0};
int motL_sign = 1;
float gear_ratio_l = 34;

//Right Encoder configuration variables
float encoderR_ticks = 48;          //Number of encoder ticks
int encoderR_sign = -1;             //Sign of the encoder velocity
int encoderR_MeasType = 1;          //Encoder velocity measurement type. 1-count pulses (brushed motor); 2-measure pulse duration (brushless motors)
int encoderR_pins[2] = {34, 36};    //Encoder Pins (Dc brushed)[EncoderA, EncoderB]; (Brushless)[Encoder, DirPin]

//Left Encoder configuration variables
float encoderL_ticks = 48;
int encoderL_sign = -1;
int encoderL_MeasType = 1;
int encoderL_pins[2] = {35, 39};

///Define functions and initialise Objects
Encoder EncR;
Encoder EncL;
MotorDriver Mr;
MotorDriver Ml;

//Define function to set up the encoders and motors
void setupMotEnc();
```

```
//Setup
void setup()
{
  //Set up the Motors and Ecoders
  setupMotEnc();
  //Begin Serial Communication
  Serial.begin(115200);
}

//Loop
void loop()
{

  EncR.ReadSensors();                       //Read the Encoders
  EncL.ReadSensors();

  Ml.MotorWrite(0.5);                       //Set Velocity percentage to the Motors (-1 to 1)
  Mr.MotorWrite(0.4);

  Serial.print(EncR.Get_Speed());           //Serial print the motor velocities
  Serial.print("\t");
  Serial.print(EncL.Get_Speed());
  Serial.print("\n");

  delay(30);                                //Delay before next loop iteration
}


void setupMotEnc()
{

  //Setup the Right Motor object
  Mr.SetBaseFreq(5000);                                     //PWM base frequency setup
  Mr.SetMotorType(motorR_type);                             //Motor type brushless, brushed setup
  Mr.SetSign(motR_sign);                                    //Set up motor sign

  if (motorR_type == BRUSHLESS_MOTOR)                       //Setup the motor pins depending on the motor tyep (brushless or brushed motor)
  {
    Mr.DriverSetup(motR_pins[0], 0, motR_pins[1]);
  }
  else if (motorR_type == DC_MOTOR)
  {
    Mr.DriverSetup(motR_pins[0], 0, motR_pins[1], motR_pins[2]);
  }

  EncR.SetTicksPerRev(encoderR_ticks);                       //Set the number of enconder pulses per revolution
  EncR.SetGearRatio(gear_ratio_r);                           //Set the gear ratio
  EncR.SetSign(encoderR_sign);                               //Set the encoder velocity sign
  EncR.SetMeasureType(encoderR_MeasType);                    //Set the encoder measurement type 1-count pulses (brushed motor); 2-measure pulse duration (brushless motors)
  EncR.SetEncType(motorR_type);                              //Set encoder motor type
  EncR.Encoder_setup(encoderR_pins[0], encoderR_pins[1], 'R');      //Set encoder pins

  Mr.MotorWrite(0);                                          //Write 0 velocity to the motor when initialising

  //Setup the Left Motor object
  Ml.SetBaseFreq(5000);
  Ml.SetMotorType(motorL_type);
  Ml.SetSign(motL_sign);
  if (motorL_type == BRUSHLESS_MOTOR)
  {
    Ml.DriverSetup(motL_pins[0], 1, motL_pins[1]);
  }
  else if (motorL_type == DC_MOTOR)
  {
    Ml.DriverSetup(motL_pins[0], 1, motL_pins[1], motL_pins[2]);
  }

  EncL.SetTicksPerRev(encoderL_ticks);
  EncL.SetGearRatio(gear_ratio_l);
  EncL.SetSign(encoderL_sign);
  EncL.SetMeasureType(encoderL_MeasType);
  EncL.SetEncType(motorL_type);
  EncL.Encoder_setup(encoderL_pins[0], encoderL_pins[1], 'L');

  Ml.MotorWrite(0);

}
```

FIGURE 21 EXAMPLE CODE FOR DC BRUSHED MOTORS USING THE MOTOR AND ENCODER LIBRARIES PROVIDED BY MANCHESTER ROBOTICS LTD

BRUSHLESS MOTOR VERSION (DEPRECATED VERSION)

```
/**
   \file Example.ino
   \author Eduard Codres, Mario Martinez
   \copyright Manchester Robotics
   \date Octiber, 2020
   \brief Example program for reaiding and writing motor velocities.
*/

//Inlcude the libraries and header files to be used, for this case
#include "Encoder.h"
#include "MotorDriver.h"


using namespace std;

//Define the global variables to configure the motors and encoders

//Right Motor Configuration Variables
int motorR_type = 1;                //Type of motor 1-brushless, 2 DC Brushed
int motorR_pins[3] = {22, 21};      //Define the Motor Pins
int motorR_sign = 1;                //Define the motor rotation sign
float gear_ratio_r = 34;            //Define gear ratio

//Left Motor configuration variables
int motorL_type = 1;
int motorL_pins[3] = {26, 25};
int motorL_sign = -1;
float gear_ratio_l = 34;

//Right Encoder configuration variables
float encoderR_ticks = 6;           //Number of encoder ticks
int encoderR_sign = -1;             //Sign of the encoder velocity
int encoderR_MeasType = 2;          //Encoder velocity measurement type. 1-count pulses (brushed motor); 2-measure pulse duration (brushless motors)
int encoderR_pins[2] = {23, 21};    //Encoder Pins (Dc brushed)[EncoderA, EncoderB]; (Brushless)[Encoder, DirPin]

//Left Encoder configuration variables
float encoderL_ticks = 6;
int encoderL_sign = 1;
int encoderL_MeasType = 2;
int encoderL_pins[2] = {27, 25};

///Define functions and initialise Objects
Encoder EncR;
Encoder EncL;
MotorDriver Mr;
MotorDriver Ml;



//Define function to set up the encoders and motors
void setupMotEnc();

//Setup
void setup()
{
  //Set up the Motors and Ecoders
  setupMotEnc();
  //Begin Serial Communication
  Serial.begin(115200);
}

//Loop
void loop()
{
  EncR.ReadSensors();                       //Read the Encoders
  EncL.ReadSensors();

  Ml.MotorWrite(0.4);                       //Set Velocity percentage to the Motors (-1 to 1)
  Mr.MotorWrite(0.4);

  Serial.print(EncR.Get_Speed());           //Serial print the motor velocities
  Serial.print("\t");
  Serial.print(EncL.Get_Speed());
  Serial.print("\n");

  delay(50);                                //Delay before next loop iteration
}
```

```
void setupMotEnc()
{

  //Setup the Right Motor object
  Mr.SetBaseFreq(5000);                                          //PWM base frequency setup
  Mr.SetMotorType(motorR_type);                                  //Motor type brushless, brushed setup
  Mr.SetSign(motorR_sign);                                       //Set up motor sign

  if (motorR_type == BRUSHLESS_MOTOR)                            //Setup the motor pins depending on the motor tyep (brushless or brushed motor)
  {
    Mr.DriverSetup(motorR_pins[0], 0, motorR_pins[1]);
  }
  else if (motorR_type == DC_MOTOR)
  {
    Mr.DriverSetup(motorR_pins[0], 0, motorR_pins[1], motorR_pins[2]);
  }

  EncR.SetEncType(motorR_type);                                   //Set encoder motor type
  EncR.SetTicksPerRev(encoderR_ticks);                            //Set the number of enconder pulses per revolution
  EncR.SetGearRatio(gear_ratio_r);                                //Set the gear ratio
  EncR.SetSign(encoderR_sign);                                    //Set the encoder velocity sign
  EncR.SetMeasureType(encoderR_MeasType);                         //Set the encoder measurement type 1-count pulses (brushed motor); 2-measure pulse duration (brushless motors)
  EncR.Encoder_setup(encoderR_pins[0], encoderR_pins[1], 'R');    //Set encoder pins

  Mr.MotorWrite(0);                                               //Write 0 velocity to the motor when initialising

  //Setup the Left Motor object
  Ml.SetBaseFreq(5000);
  Ml.SetMotorType(motorL_type);
  Ml.SetSign(motorL_sign);
  if (motorL_type == BRUSHLESS_MOTOR)
  {
    Ml.DriverSetup(motorL_pins[0], 1, motorL_pins[1]);
  }
  else if (motorL_type == DC_MOTOR)
  {
    Ml.DriverSetup(motorL_pins[0], 1, motorL_pins[1], motorL_pins[2]);
  }

  EncL.SetEncType(motorL_type);
  EncL.SetTicksPerRev(encoderL_ticks);
  EncL.SetGearRatio(gear_ratio_l);
  EncL.SetSign(encoderL_sign);
  EncL.SetMeasureType(encoderL_MeasType);
  EncL.Encoder_setup(encoderL_pins[0], encoderL_pins[1], 'L');

  Ml.MotorWrite(0);

}
```

FIGURE 22 EXAMPLE CODE FOR BRUSHLESS MOTORS USING THE MOTOR AND ENCODER LIBRARIES PROVIDED BY MANCHESTER ROBOTICS LTD

UPLOADING THE PROGRAM TO PUZZLE-BOT

To upload a program to the ESP32 microcontroller follow the next steps.

1. Connect the microUSB cable to the Puzzle-Bot Control Module. Make sure the Main Switch is ON and the Power Selector Switch is in USB mode. Windows will automatically install the drivers for using the microcontroller, in case windows does not install the drivers, please refer to the next section.

Note that the figure is only for illustrative purposes, you don't have to detach the Control Module from the robot.

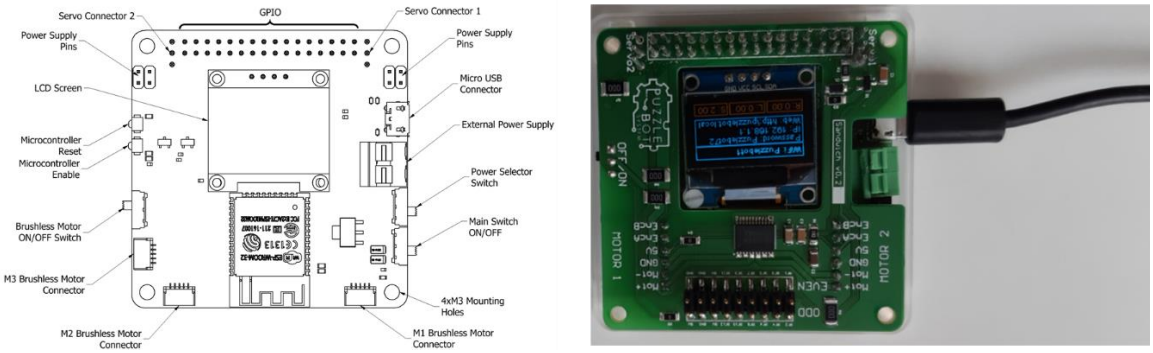


FIGURE 23 USB CONNECTION TO PUZZLE-BOT CONTROL MODULE

2. Select the ESP32 Dev Module from Tools>Board>ESP32 Dev Module

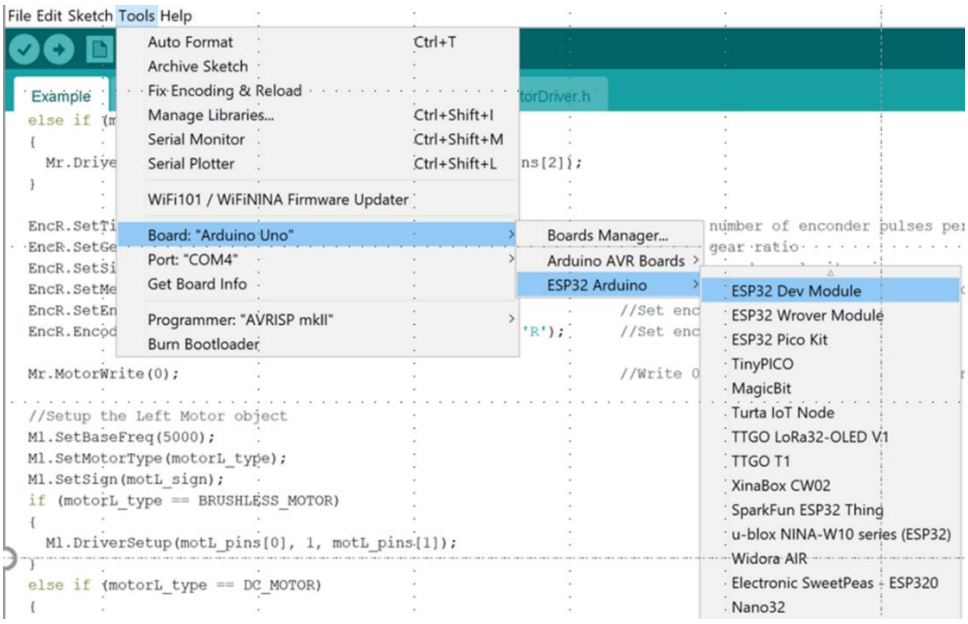


FIGURE 24 ESP32 BOARD SELECTION

3. Select the computer USB port (COM) where the Control Module is connected (In case Arduino does not recognize the port go to the next section).

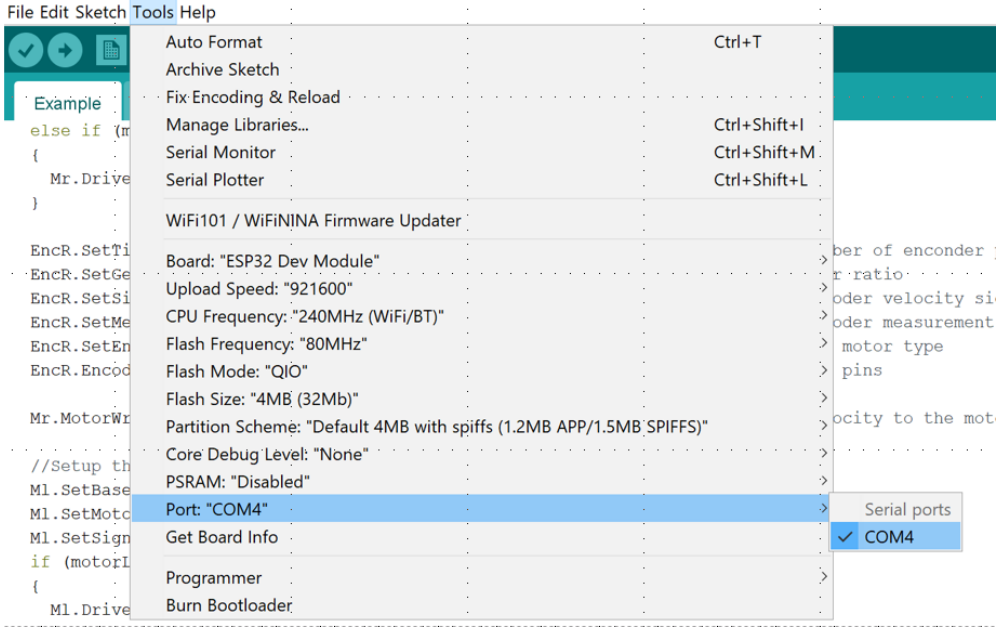


FIGURE 25 USB PORT (COM) SELECTION

4. Upload the program to the Puzzle-Bot Control Module.

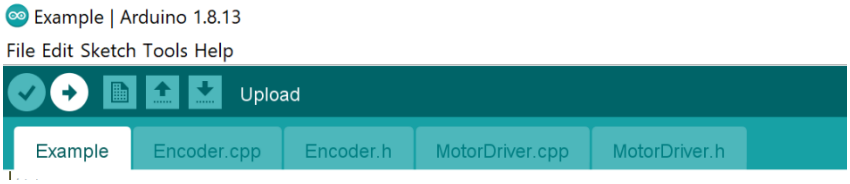


FIGURE 26 UPLOAD THE PROGRAM TO THE PUZZLE-BOT CONTROL MODULE

5. Wait until the program finish uploading.

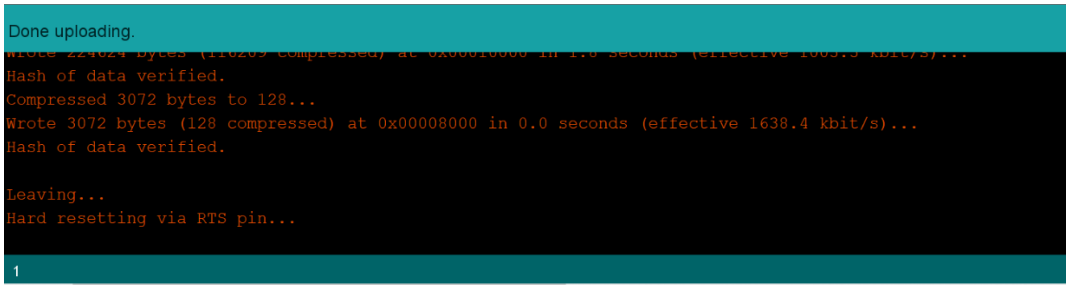


FIGURE 27 PROGRAM DONE UPLOADING

FAQ

1. How do I know if the drivers are properly installed?
   Plug the Puzzle-Bot into the USB port
   Go to Start > Device Manager
   The Serial port should appear as shown in the following figure (The COM port may vary).

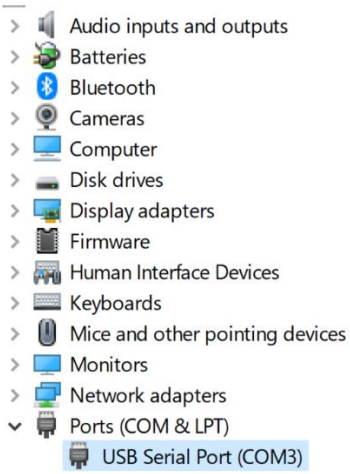


FIGURE 28 COM PORT

2. My computer could not find the drivers
   Download the drivers from the following link
   https://ftdichip.com/drivers/vcp-drivers/
   Scroll down and download the executable setup as shown in the following figure

Operating System	Release Date	X86 (32-Bit)	X64 (64-Bit)	PPC	ARM	MIPSII	MIPSIV	SH4	Comments
Windows*	2021-07-15	2.12.36.4	2.12.36.4	–	–	–	–	–	WHQL Certified. Includes VCP and D2XX. Available as a setup executable ← Please read the Release Notes and Installation Guides.
Linux	–	–	1.5.0	–	–	–	–	–	All FTDI devices now supported in Ubuntu 11.10, kernel 3.0.0-19 Refer to TN-101 if you need a custom VCP VID/PID in Linux VCP drivers are integrated into the kernel.
Mac OS X 10.3 to 10.8	2012-08-10	2.2.18	2.2.18	2.2.18	–	–	–	–	Refer to TN-105 if you need a custom VCP VID/PID in MAC OS
Mac OS X 10.9 to 10.14	2019–12–24	–	2.4.4	–	–	–	–	–	This driver is signed by Apple

FIGURE 29 DRIVERS DOWNLOAD

   Unplug the Puzzle-Bot from the computer.
   Unzip the drivers and run the setup (some computers are required to be restarted after the installation).
   Plug the Puzzle-Bot back into the computer and go to Q1.

3. My computer still not recognize the drivers even after the installation
   Plug the Puzzle-Bot into the USB port.

Go to Start > Device Manager.
Look for the USB Serial Converter as shown in the following picture.

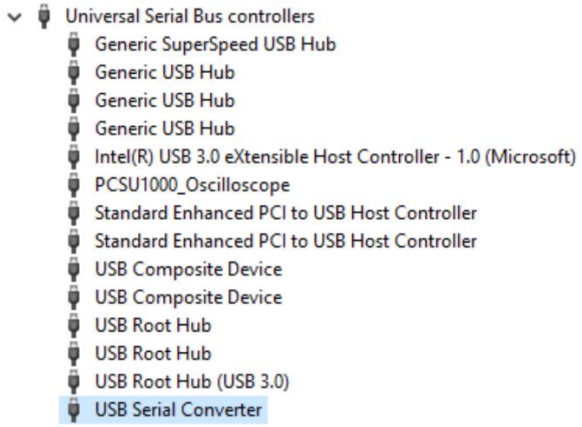


FIGURE 30 USB SERIAL CONVERTER

Right Click to Properties > Advanced Tab.
Make sure the Load VCP box is checked.
Reconnect the Puzzle-Bot to the computer.

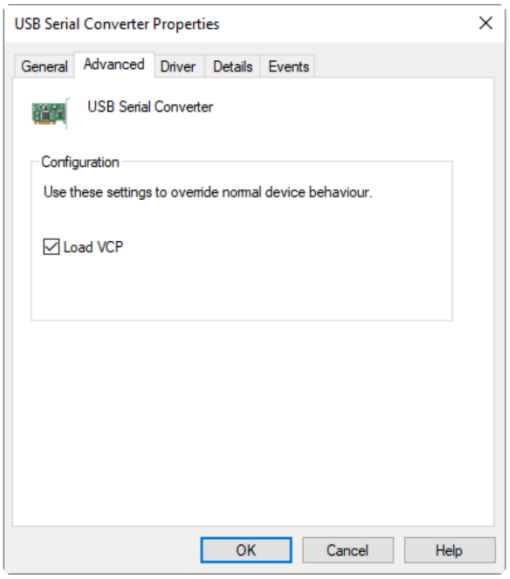


FIGURE 31 VCP PORT

GETTING STARTED (TURNING ON THE PUZZLE-BOT)

STARTING THE PUZZLE-BOT

BRUSHLESS MOTOR VERSION

1. Connect the Power Bank (if included) to the Micro USB connector or leave the computer connected to the robot or a regulated Power Supply (Battery etc.) to the external Power supply connector.
2. Select the Power supply from the power selector switch (Battery/USB).
3. Turn ON the Brushless Motor Switch.
4. Turn ON the Main Switch.
5. Your program should be running now.

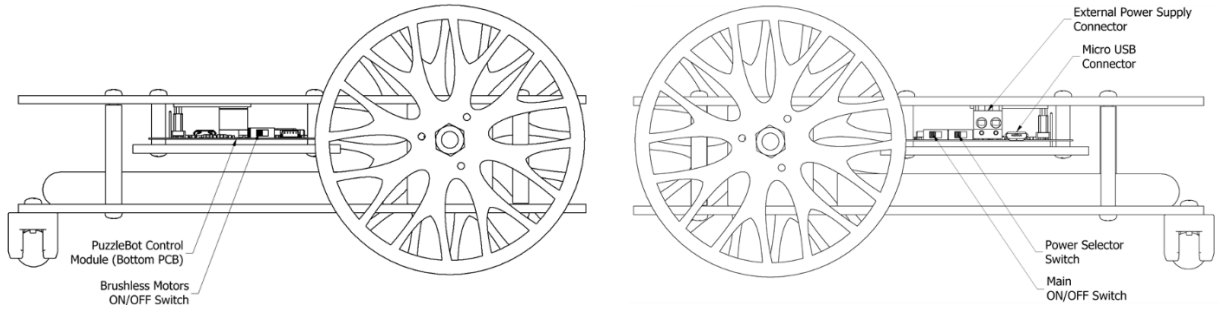


FIGURE 32 PUZZLE-BOT BRUSHLESS MOTOR VERSION, SIDE VIEW (RIGHT, LEFT RESPECTIVELY)

BRUSHED MOTOR VERSION

1. Follow Steps 1 and 2 from the Brushless Motor Version.
2. Turn ON the DC Brushed Motors Switch in the Top PCB.
3. Turn ON the Main Switch.
4. Your program should be running now.

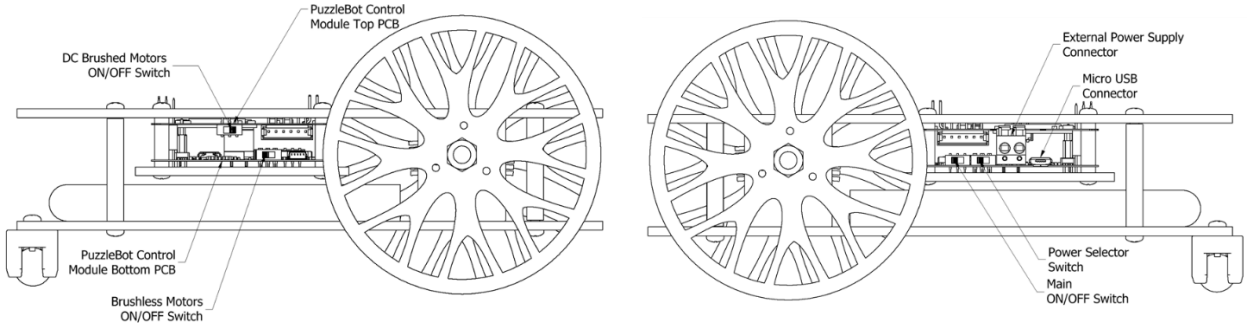


FIGURE 33 PUZZLE-BOT DC BRUSHED MOTOR VERSION, SIDE VIEW (RIGHT, LEFT RESPECTIVELY)

GETTING STARTED (EXTERNAL COMPUTING CONFIGURATION)

In the following section a basic introduction on how to get started with the robot using the external computing configuration will be presented. Some basic functionalities, such as starting the Puzzle-Bot, connecting to the robot, basic testing and parameter configuration will be shown.

Note: For this configuration, the binaries are preloaded, if not refer to the section How to flash.

STARTING THE PUZZLE-BOT

BRUSHLESS MOTOR VERSION

6. Connect the Power Bank (if included) to the Micro USB connector or a regulated Power Supply (Battery etc.) to the external Power supply connector.
7. Select the Power supply from the power selector switch (Battery/USB).
8. Turn ON the Brushless Motor Switch.
9. Turn ON the Main Switch.
10. Verify that the screen is displaying information as shown in Figure 36.

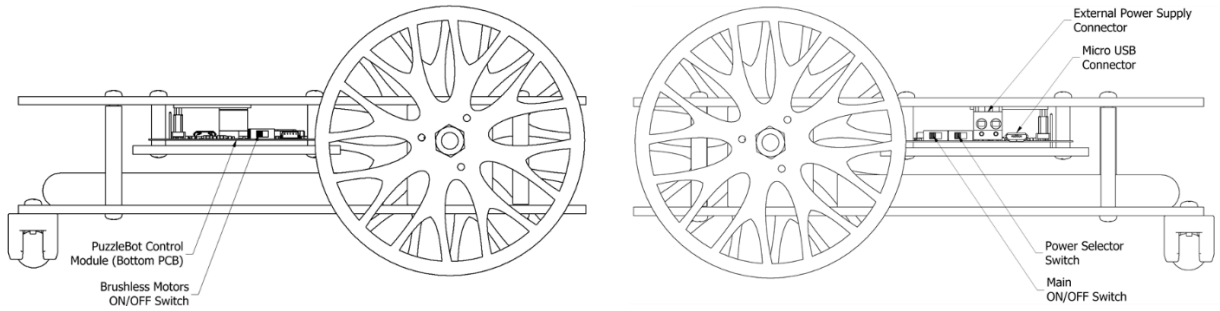


FIGURE 34 PUZZLE-BOT BRUSHLESS MOTOR VERSION, SIDE VIEW (RIGHT, LEFT RESPECTIVELY)

BRUSHED MOTOR VERSION

5. Follow Steps 1 and 2 from the Brushless Motor Version.
6. Turn ON the DC Brushed Motors Switch in the Top PCB.
7. Turn ON the Main Switch.
8. Verify that the screen is displaying information as shown in Figure 36.

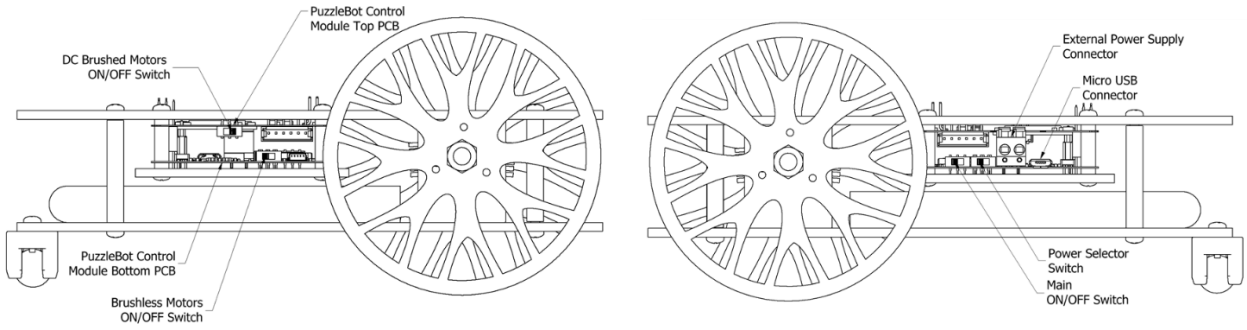


FIGURE 35 PUZZLE-BOT DC BRUSHED MOTOR VERSION, SIDE VIEW (RIGHT, LEFT RESPECTIVELY)

FIGURE 36 INFORMATION DISPLAYED ON THE LCD SCREEN (SSID, IP AND PASSWORD MAY DIFFER DEPENDING ON THE CONFIGURATION).

CONNECTING TO THE PUZZLE-BOT

The following sections are used for the **External Computing Unit Configuration**, using the firmware binaries provided by Manchester Robotics ltd. Please note that all the robots come pre-programmed with such binaries unless the user modify the program (On Board Configuration). The original binaries can be flashed anytime by following the steps in the section (How to Flash).

1. Select the Network (📶) icon on the taskbar. The icon that appears depends on your current connection state. If you don't see one of the network icons (or a similar one) shown in the following image, select the Up arrow (^) to see if it appears there.
2. Choose the Puzzle-Bot Wi-Fi network of the robot you want (Puzzle-Bot x), then select Connect.
3. Type the network password as shown on the LCD Display, and then select Next.

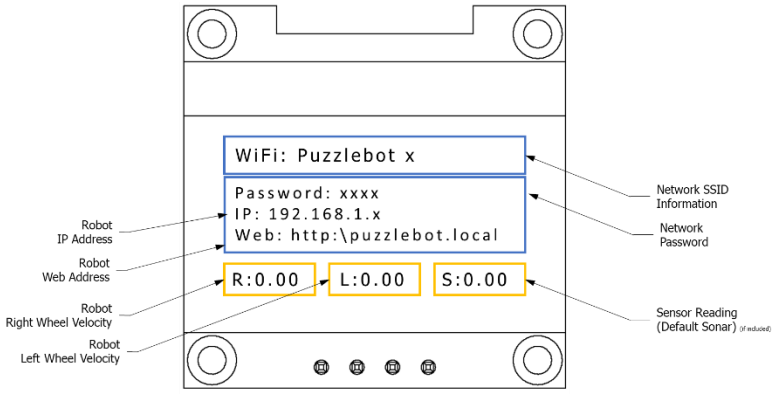


FIGURE 37 LCD SCREEN INFORMATION

4. Once Connected the following figure in the Network Settings should appear.

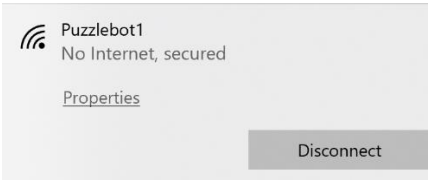


FIGURE 38 SUCCESSFUL CONNECTION TO PUZZLE-BOT

BASIC WEB INTERFACE

The robot has a web interface when in external computing unit configuration; for testing the different sensors and actuators equipped in the robot. In this webpage the speeds of the motors can be set, to be tested and driven to a desired speed. The measured speed of the wheels will be shown on the interface, as well as the data from any sensors attached to the robot.

ACCESSING THE WEB INTERFACE

1. Open any web browser

FIGURE 39 WEB BROWSER EXAMPLES

2. Type in the search bar (address bar) the IP Address of the robot as displayed on the LCD screen in Figure 37.

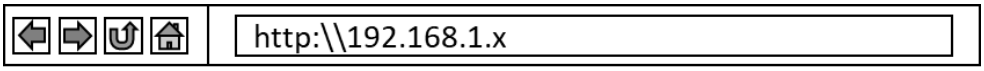


FIGURE 40 ADDRESS BAR

3. The following webpage should appear.

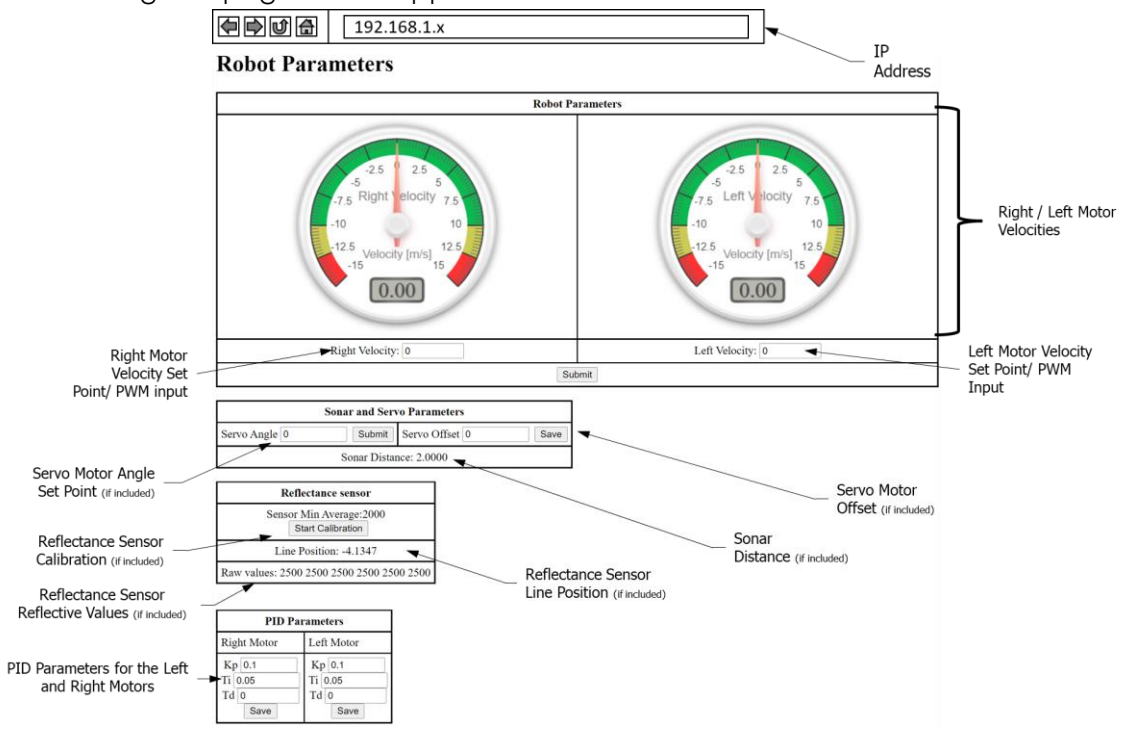


FIGURE 41 ROBOT WEB INTERFACE

TABLE 3 WEB INTERFACE GENERAL DESCRIPTION

IP Address	IP address of the robot of control
Right/Left Motor Velocities	Right/ Left wheel velocities display
Right/Left Motor Velocity Set Point	Set Point for the right and left wheel velocities.
Servo Motor Offset*	Offset of the servo motor
Servo Motor Angle Set Point*	Set Point for the servo motor angle
Sonar Distance**	Sonar distance display
Reflectance Sensor Calibration***	Initialise calibration of the reflectance sensor
Reflectance Sensor Reflective Values***	Values given by each reflectance sensor $[S_1,S_2,...,S_N]$.
Reflectance Sensor Line Position***	Position of the line after calibration with respect to the reflectance sensor centre.
PID Parameters	Values of the PID control parameters for each motor.

*For information regarding Servo Motor refer to the Appendix.
**For information about sonar refer to the Appendix.
***For Reflectance sensor information and calibration refer to the Appendix.

CONFIGURATION FILE (YAML FILE)

Puzzle-Bot contains a configuration file for defining the different working modes, controllers, sensor and actuators. The configuration file is based on a YAML parameter file. To access and modify such configuration file follow steps shown in this section.

ACCESS TO THE PARAMETER FILE

1. Connect to the robot following the steps in *Connecting to the Puzzle-Bot*
2. Type in the address bar of web browser the IP address of the robot followed by "/config" as shown in the following image.

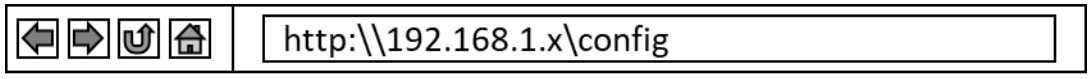


FIGURE 42 ACCESS TO THE PUZZLE-BOT CONFIGURATION FILE

3. The following page should appear.

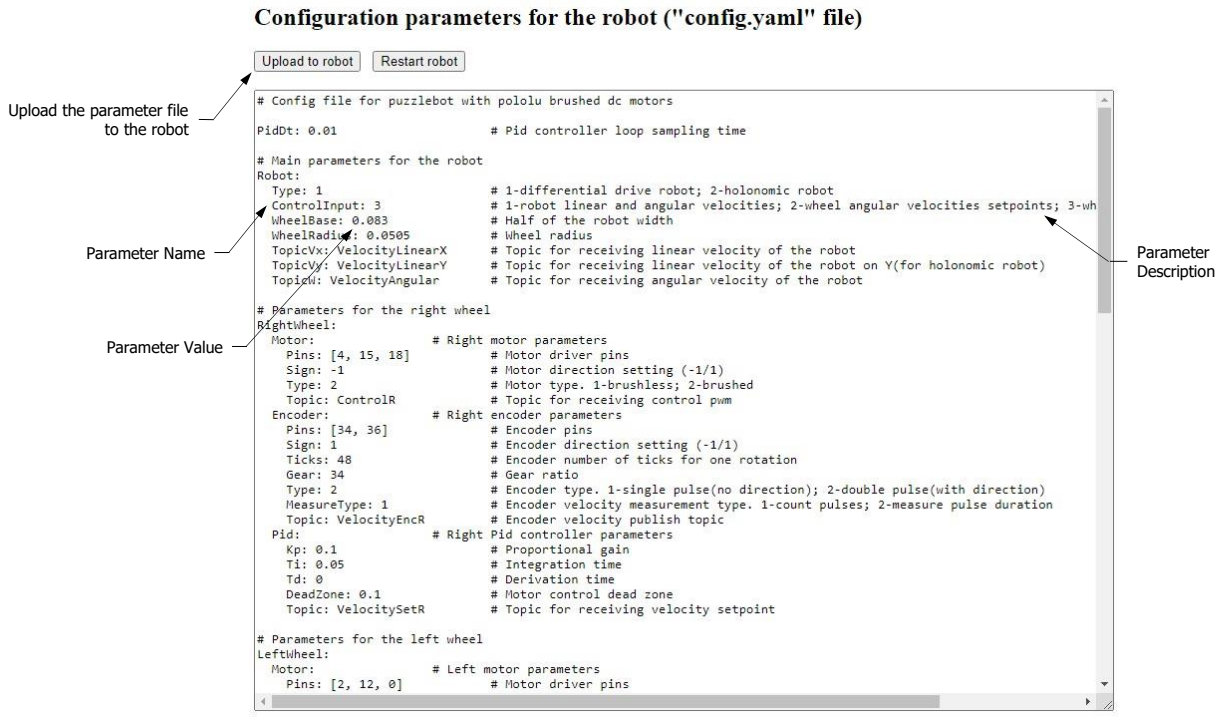


FIGURE 43 PARAMETER FILE (YAML)

4. Manchester Robotics ltd. facilitates the users previously defined parameter configuration files for both Brushless (Deprecated) and Brushed DC motor Puzzle-Bot versions. They can be found in the folder Binaries >> config_files and easily used by following the next steps.
    I. Open the configuration webpage as defined in steps 1 to 3.
    II. Download pre-defined parameter configuration files for the desired Puzzle-Bot Version.

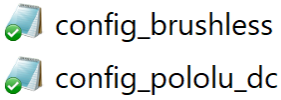


FIGURE 44 PRE-DEFINED PARAMETER CONFIGURATION FILES

III. Delete the current parameter on the webpage.
IV. Open the desired file and copy the parameters in the readme.
V. Paste the copied parameters file to the webpage shown in Figure 43.
VI. Press the "Upload to robot" button at the top of the webpage as shown in Figure 43.
VII. Restart the robot by turning it OFF and ON again from the main switch.

PARAMETER CONFIGURATION FILE DESCRIPTION

As stated before, when using the binaries provided by Manchester Robotics ltd, the robot can be configured to work in different modes and with different sensors and actuators. In the following sections the basic working modes that are configured in the parameter configuration file will be described.

MAIN PARAMETERS OF THE ROBOT

In this section of the parameter configuration file, the main parameters that describe the type of robot and its physical characteristics are described.

- The type of robot describes the robot to be used as differential drive robot(1) or a holonomic robot (2). <u>Note</u>, *Puzzle-Bot is a differential drive robot.*
- Changing this parameter (1,2 or 3) allows Puzzle-Bot to work with different control inputs depending on the user.
    1. Robot Linear and Angular Velocities provide the linear v and angular ω velocities to the robot as seen in Figure 45. The velocities are then converted in the firmware to wheel velocities and a PID controller in each wheel, makes the robot reach its set point (PID control parameters for each wheel can be re-tuned as explained in the following section).

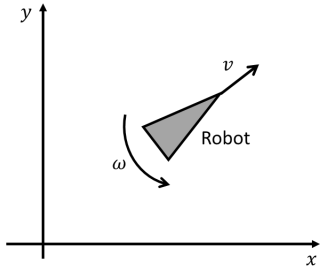


FIGURE 45 ROBOT VELOCITIES

    2. Wheel angular velocities, form the external computing unit, the robot receives as inputs the set points for each wheel angular velocities (ω_r, ω_l). As in the previous case, a PID controller, inside the firmware, is used to regulate the speed of each wheel. PID parameters can be re-tuned in the following section.
    3. Wheel PWM voltage signals, this input mode receives a value between [-1, 1], from the external computing unit, for setting the wheel angular speed. The interval [-1,1] maps the duty cycle of the signal [0%, 100%]and the wheel rotation direction i.e. $[-1{,}1] \rightarrow [-100\%, 100\%]$, where -100% represents full power backwards and 100% full power forward. Note that this configuration does not run any controller in the firmware.
- Wheelbase and Wheel Radius represent the physical characteristics of the robot used for transforming the robot velocities (v, ω) into wheel angular velocities (ω_r, ω_l). Wheelbase is half of the distance between wheels of the robot (robot width) and wheel radius is the radius of the wheels.
- Topic Vx, Vy, W, used for receiving linear and angular velocities from an external computing unit. Further information, refer to the advanced manual configuration.

TABLE 4 MAIN PARAMETERS OF THE ROBOT

Main parameters for the robot		
Robot:		
Type:	1	1-differential drive robot; 2-holonomic robot
ControlInput:	3	1-robot linear and angular velocities; 2-wheel angular velocities setpoints; 3-wheel PWM signals
WheelBase:	0.083	Half of the robot width
WheelRadius:	0.0505	Wheel radius
TopicVx:	VelocityLinearX	Topic for receiving linear velocity of the robot**
TopicVy:	VelocityLinearY	Topic for receiving linear velocity of the robot on Y(for holonomic robot)**
TopicW:	VelocityAngular	Topic for receiving angular velocity of the robot**

**For further information regarding topics for controlling the robot, refer to the advanced manual configuration (Advanced Users).

PARAMETERS OF THE WHEELS (LEFT AND RIGHT)

In this section of the parameter configuration file, the main parameters that describe the type of motors and encoders used for the right and left wheels and its physical parameters are described. *For the purposes of simplicity only one table will be shown nonetheless please note that the parameters for the right and left motors are defined separately.*

- Motor Pins, define the microcontroller pins used to connect the motor. For the Brushed version of the Puzzle-Bot, the pin vector represents [PWM, InA, InB] respectively. For the brushless motor version, the pin vector represents [PWM, Dir], respectively. Refer to Pinout reference for further information.
- Motor Sign, motor rotation direction sign setting.
- Motor Type defines the type of motor to be used, i.e., brushed or brushless.
- Motor Topic defines the topic used for receiving the PWM signal. Further information, refer to the advanced manual configuration.
- Encoder Pins, define microcontroller pins used to connect the encoders. For the Brushed version of the Puzzle-Bot pin vector, represents [EncB, EncA], respectively, for the brushless motor version the pin vector represents [Enc, Dir], respectively. Refer to Pinout reference and motor datasheets for further information.
- Encoder Sign, define the encoder direction setting.
- Encoder Ticks, number of ticks from the encoder per rotation.
- Encoder Gear, define the gear ratio of the motor as x: 1 where x is the input ratio.
- Encoder Type, define the type of encoder (1) single channel, (2) dual channel encoder. Note Brushed motors contain a double channel encoder, brushless motors are single channel.
- Encoder Measure Type, Define the way of measurement for the different type of encoders. Dual channel, count pulses (2), Single channel, pulse duration (1).
- Encoder Topic defines the topic used for publishing the velocity measured by the encoders. Further information, refer to the advanced manual configuration.

- PID Kp, define the proportional gain for the left/right wheel PID controller.
- PID Ti, define the Integration time for the left/right wheel PID controller.
- PID Td, define the Derivation time for the left/right wheel PID controller.
- PID Dead Zone, define the dead zone symmetrical around zero, for the left/right wheel PID controller.
- PID Topic, defines the topic used for receiving the velocity set point defined by the user. Further information refer to the advanced manual configuration.

TABLE 5 CONFIGURATION PARAMETERS FOR THE LEFT/RIGHT WHEEL

Parameters for the right/left wheel		
Right/Left Wheel:		
Motor:		Right motor parameters
Pins:	[4, 15, 18]	Motor driver pins
Sign:	1	Motor direction setting (-1/1)
Type:	2	Motor type. 1-brushless; 2-brushed
Topic:	ControlR	Topic for receiving control pwm**
Encoder:		Right encoder parameters
Pins:	[36, 34]	Encoder pins
Sign:	-1	Encoder direction setting (-1/1)
Ticks:	48	Encoder number of ticks for one rotation
Gear:	34	Gear ratio
Type: 2	2	Encoder type. 1-single pulse(no direction); 2-double pulse(with direction)
MeasureType:	1	Encoder velocity measurement type. 1-count pulses; 2-measure pulse duration
Topic:	VelocityEncR	Encoder velocity publish topic**
Pid:		Right Pid controller parameters
Kp:	0.1	Proportional gain
Ti:	0.05	Integration time
Td:	0	Derivation time
DeadZone:	0.1	Motor control dead zone
Topic:	VelocitySetR	Topic for receiving velocity setpoint**

**For further information regarding topics for controlling the robot, refer to the advanced manual configuration (Advanced Users).

NETWORK PARAMETERS

In this section of the parameter configuration file, the main Network parameters are described.

- Nework SSID, Sets the Network name or "access point ID", *shown in the LCD display*.
- Nework Password, Sets the Network Password, *shown in the LCD display*.
- Nework IP, Sets the IP of the Access Point, *shown in the LCD display*.
- Nework RosIP, Sets the IP for the ROS master, when ROS connectivity is enabled.
- Nework PortWeb, Sets the web server port.

- Nework PortData, Sets the Port for data communication server (tcp/udp).
- Nework WiFi Power, Sets the WiFi transmitting power in the ESP32.
- Nework WebPage, Sets the Domain name of the webpage.

TABLE 6 NETWORK CONFIGURATION PARAMETERS

Network parameters		
Network:		
SSID:	Puzzlebot1	Access point ssid
Password:	Puzzlebot72	Access point password
IP:	[192, 168, 1, 1]	Access point IP
RosIP:	[192, 168, 1, 2]	Ros master IP (when ros connection is active)**
PortWeb:	80	Port for the web server**
PortData:	3141	Port for data communication server (tcp/udp) **
WiFiPower:	78	Wifi transmitter power (78 is maximum)
WebPage:	puzzlebot	Domain name for the webpage

**For further information regarding server configuration and ROS connectivity for controlling the robot, refer to the advanced manual configuration (Advanced Users).

Active Modules Configuration

In this section of the parameter configuration file, the active module configuration will be described. Active modules refer to the modules that come pre-programmed by Manchester Robotics ltd. in the Puzzle-Bot, these can be activated or deactivated if the modules are included in the Puzzle-Bot system.

- Active Modules: Servo Motor, Sonar, Reflectance Sensor, Screen, ROS, Activate the modules by setting a 1 for activating and a 0 for deactivating the modules in Puzzle-Bot.

TABLE 7 ACTIVE MODULES CONFIGURATION

Active modules settings		
ActiveModules:		
Servo:	1	Servo motor on/off (1/0)
Sonar:	1	Sonar on/off (1/0)
Reflect:	1	Reflectance sensorr on/off (1/0)
Screen:	1	Screen on/off (1/0)
Ros:	0	Ros communication on/off (1/0)**

**For further information regarding ROS connectivity for controlling the robot, refer to the advanced manual configuration (Advanced Users).

Sensor and Actuator Configuration

In this section the parameter configuration for the Sonar, Servo Motor and Reflectance sensor (if included), will be described.

- Sonar Pins, Input pin of the sonar, refer to pinout diagram section.

- Sonar Topic, define the Topic for publishing sonar measured distance. For further information, refer to the advanced manual configuration.
- Servo Pins, define the Servo motor pin.
- Servo Offset, define the Servo offset from the center.
- Servo Topic, defines the topic used for receiving the servo motor angle set point (deg) defined by the user. For further information, refer to the advanced manual configuration.
- Reflectance NPins, Define the number of reflectance sensors used.
- Reflectance Pins, Define Reflectance sensor input pins in the Puzzle-Bot Control Unit [S_1, S_2,…, S_{NPins}].
- Reflectance EmitterPins, define the emitter pins of the reflectance sensor, refer to the datasheet of the reflectance sensor and the Pinout diagram section.
- Reflectance EmitterSelect, select the odd (1), even (2) or all (3) emitters of the reflectance sensor.
- Reflectance MinValues, define the Minimal reflectance values measured during calibration (use webpage).
- Reflectance MinAvg, set the minimal reflectance average value measured during calibration
- Reflectance Threshold, line detection threshold.
- Reflectance Topic, set the topic used for publishing the line position when using the reflectance sensor, with respect to the center of the sensor. Further information, refer to the advanced manual configuration.

TABLE 8 SENSOR AND ACTUATOR CONFIGURATION

Parameters for sonar		
Sonar:		
Pins:	14	Sonar pins
Topic:	SonarDistance	Topic for publishing sonar measured distance**
Parameters for servo motor		
Servo:		
Pins:	5	Servo motor pins
Offset:	0	Servo motor center offset
Topic:	ServoAngle	Topic for receiving servo motor angle (degrees)**
Parameters for reflectance sensor		
Reflectance:		
NPins:	6	Number of pins (number of light sensors used)
Pins:	[32, 25, 27, 19, 23, 22]	Reflectance pins
EmitterPins:	[13, 12]	Emitter pins (for selecting even/odd lights on/off)
EmitterSelect:	2	Emitter selector. 1-even; 2-odd; 3-all

MinValues:	[600, 600, 600, 600, 600, 600]	Minimal reflectance values measured during calibration (use webpage)
MinAvg:	2000	Minimal reflectance average value measured during calibration
Threshold:	1.1	Line detection threshold
Topic:	LineSensor	Topic for publishing line position relative to the sensor**

**For further information regarding functionality of the sensors and actuators and topics for controlling or reading them, refer to the advanced manual configuration (Advanced Users).

HOW TO FLASH

In this section a way to flash the program binaries into the Puzzle-Bot along with the configuration files will be presented.

WINDOWS

1. Attach the microUSB cable to the Puzzle-Bot Control Module. *Note the image is just for explanation purposes, detachment of the Puzzle-Bot Control module from the robot is not necessary.*

FIGURE 46 PUZZLE BOT CONTROL UNIT WITH MICRO USB

2. Make sure in the Bottom PCB that the Power Selector Switch is in the USB Position, and the Main Switch is turned ON.

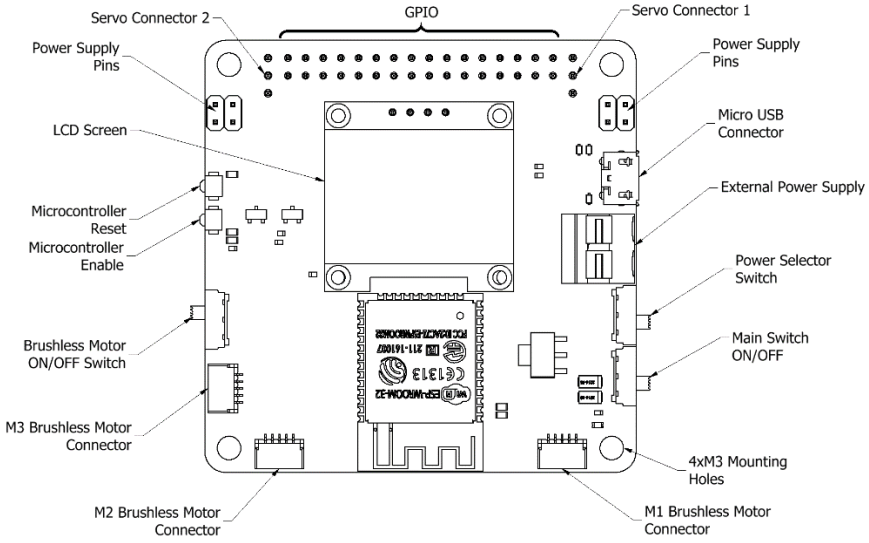


FIGURE 47 PUZZLE-BOT BOTTOM PCB

3. Connect the USB to any free USB port in the computer

4. Decompress the Firmware files "FirmwareBin-1.x.x.zip" into a folder.

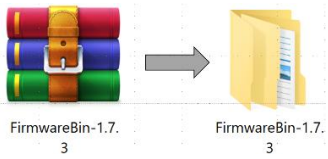


FIGURE 48 DECOMPRESS FIRMWARE FILE

5. Open the folder and select the corresponding operating system.

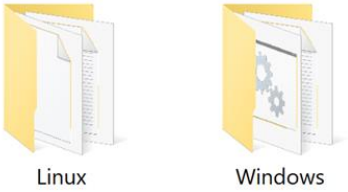


FIGURE 49 OPERATING SYSTEM SELECTION

6. Run FirmwareFlash.bat to flash the firmware to the esp32, the following screen should appear. Once finished press any key to exit.

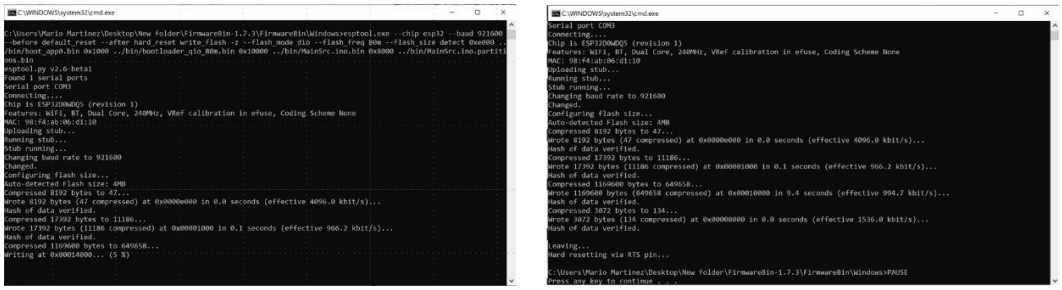


FIGURE 50 UPLOADING FIRMWARE, FLASHING SCREENS (START AND FINISH)

7. Run DataFlash.bat to flash the data files to the esp32, the following screen should appear. Once finished press any key to exit.

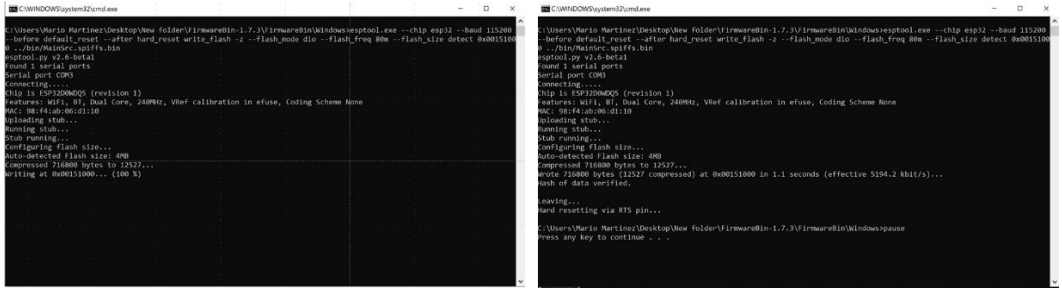


FIGURE 51 UPLOADING DATA FILES TO MICROCONTROLLER

8. After finishing uploading the firmware and the data files the LCD display should start displaying information as seen in Figure 52.

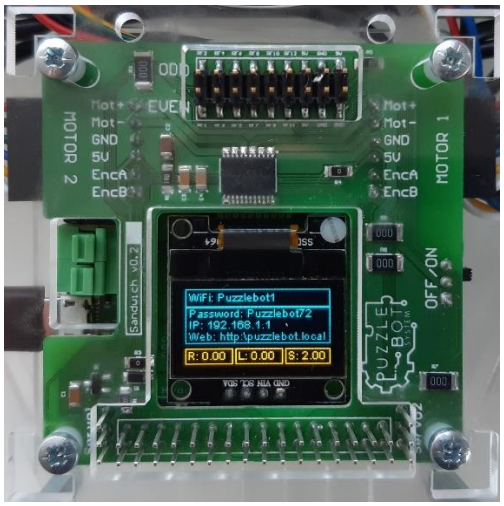


FIGURE 52 INFORMATION BEING DISPLAYED BY THE LCD DISPLAY, AFTER FLASHING

9. PuzzeBot is ready to be used. Refer to the previous section on how to get started with the Puzzle-Bot. ***Please note that the configuration parameter file (YAML) inside the microcontroller still need to be configured, for using it with the Brushless or Brushed DC motor Version please refer to the section Configuration File (YAML file).***

MATLAB ROBOTIC USER INTERFACE (MRUI)

One of the main features of the Puzzle-Bot is the capability of controlling it remotely using MATLAB. Manchester Robotics ltd. provide for this a *MATLAB Robotic User Interface* (MRUI) that includes the necessary libraries for simulating the Puzzle-Bot in its basic configuration and with the included sensors and actuators such as sonar, reflectance sensors and servo motors. At the same time, it provides the user the ability to connect and seamlessly test their own programs directly with the Puzzle-Bot.

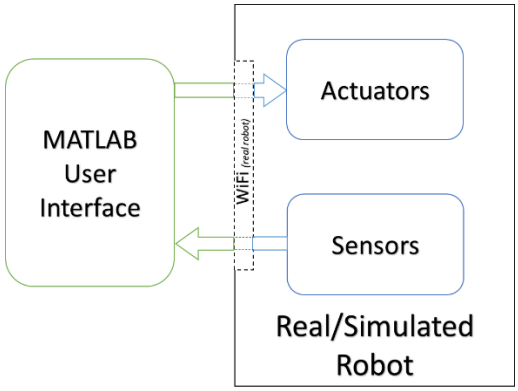


FIGURE 53 MATLAB ROBOTIC USER INTERFACE (MRUI)

MATLAB Robotic User Interface by *Manchester Robotics ltd.* can be used in a multiple variety of ways depending on the user needs. For basic usage, two main ways (Stand Alone MRUI and Arduino style MRUI) of using the *MATLAB Robotic User Interface* will be shown in the subsequent sections.

WIFI COMMUNICATION

In this section, some basic information and set up when working with the real robot is described.

- The MRUI communication with the real robot is based on UDP WiFi protocol.
- The user might experience some communication lag because of WiFi network overload.
- Make sure that no other software interferes with the network (browser, etc.)
- Close the webpage of the robot (don't use them in parallel).
- Try keeping a Line of sight between the robot and the computer (preferable).
- When implementing real time controllers, keep in mind, that the latency between MATLAB and the microcontroller is around 20 ms depending on the computer performance.

STAND ALONE STYLE (SA-MRUI)

This is the most basic configuration available, in this mode the user uses the libraries available in MATLAB created by *Manchester Robotics ltd.* to connect to the real robot or the simulated robot and send/receive different signals from the real/simulated actuators or sensors. The user has the complete flexibility offered by MATLAB to set up its own variables, control loop timing and control strategy. The diagram shown in Figure 54 depicts the Stand-Alone Style of programming for a simple control loop. Some Examples on how to code in this mode will be shown in the subsequent sections.

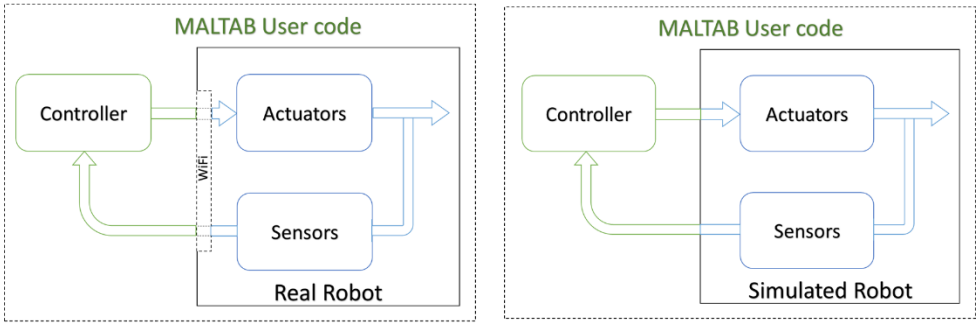


FIGURE 54 STAND ALONE STYLE FOR A SIMPE CONTROL LOOP, A)PUZZLE-BOT, B)SIMULATED ROBOT

BASIC WORKFLOW OF THE STAND-ALONE PROGRAMMING STYLE

As stated before, in the stand-alone style of programming the MRUI, the user gets the flexibility of MATLAB programming by using the libraries provided by Manchester Robotics ltd. One of the most common ways of doing this is by following a simple control algorithm workflow as the one shown in the following figure.

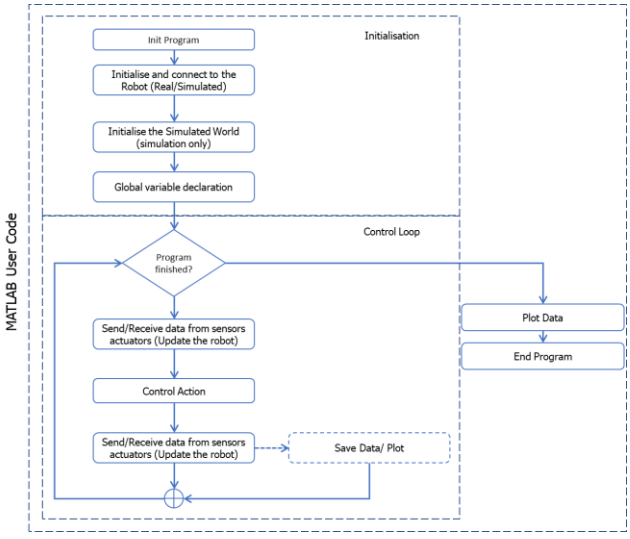


FIGURE 55 BASIC WORKFLOW OF THE SA-MRUI METHOD

INITIALISATION

- *Program Initialisation*- In this section the user initializes the program by clearing all the variables, make sure MATLAB is properly Set Up, and the path to the libraries is properly defined.
- *Initialise and connect the robot*- This section can be further divided into two subsections for real and simulated robots.
    - Real Robot- Connect to the real robot using the libraries to set the IP address of the robot and the name of the robot to be connected.
    - Simulated Robot- In this part a simulated robot is created via a JSON file that contains all the characteristics of the robot sensors and actuators as well as the variable definitions for each one of them to send or receive information. Further information on JSON files for robot designing in the subsequent sections. Manchester Robotics ltd. provides some pre-defined JSON files for the Puzzle-Bot System that contains the Motors, Encoders, servo motor and exteroceptive sensors such as Reflectance sensors and sonar.
- *Initialise the World*- In this section the user defines the world (2D-two dimensional world) where the robot is presented. This world can contain different obstacles defined by the user and it can be used when simulating LIDAR's, Sonars and Line following sensors. The World, as the robot simulation is presented, in a JSON file that contains all the characteristics of the world such as coordinate system, obstacles, colour of obstacles etc. Further information about the World JSON files will be available in the subsequent sections. Manchester Robotics ltd. provides the users with some pre-defined Worlds, empty worlds for basic simulations and worlds with obstacles and lines for line following tasks.
- *Global Variables Declaration*- In this section the user can define and initialise the variables to be used in the code. Manchester Robotics ltd. will present some examples in the subsequent sections.

CONTROL LOOP

**Note that for the Purpose of a basic explanation a control loop is defined, the user can change this loop or redefine the programming method of the robot.*

- *Program Finished*- In this section the user can establish any constrain for the control loop to finish. Some basic constrains can be time based or sensor based, etc.
- *Receive Data from Sensors*- In this section the user can get the data from the different sensors used in the robot such as encoders, sonar, reflectance etc. The variables where this information is stored depends on the JSON file for a particular simulated robot. The variables used for the real robot are given in the next section. Manchester Robotics ltd provides some examples on how to access these variables in the next sections. *Note for simplicity the name of the variables in the JSON file (default) and the variables for robot communication are the same.*
- *Control Action*- In this section the user can program the control action using the information received by the sensors in the previous point.
- *Send Data to Actuators*- As with the sensors, in this section the user is able to send the control action to the actuators using some pre-defined variables; for the simulation these variables are defined in the JSON files, and for the real robot these variables will be

provided in the following section. *Note for simplicity the name of the variables in the JSON file and the variable s for robot communication are the same.*

- *Save Data/ Plot*- In this section the user is able to save the data for future use or plot the data in real time *(keep in mind that plotting can take some time when dealing with real time control control)*.
- *Plot Data*-In this section the user can plot data if desired for visualisation.
- *End Program*- In this section the program ends.

GETTING STARTED STAND ALONE STYLE (SA-MRUI)

As stated before, this style of controlling the robot is based on the libraries provided by Manchester Robotics ltd. To start controlling the robot in this way the following steps must be followed.

SETTING UP THE MATLAB CONTROL ENVIRONMENT

1. Unzip the MATLAB Robotic User Interface (*mobile-robot-matlab-ui.rar*)

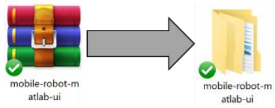


FIGURE 56 UNZIP THE MATLAB ROBOTICS USER INTERFACE

2. The MRUI folder contains different files as shown in Figure 57, each file will be explained in the subsequent sections.

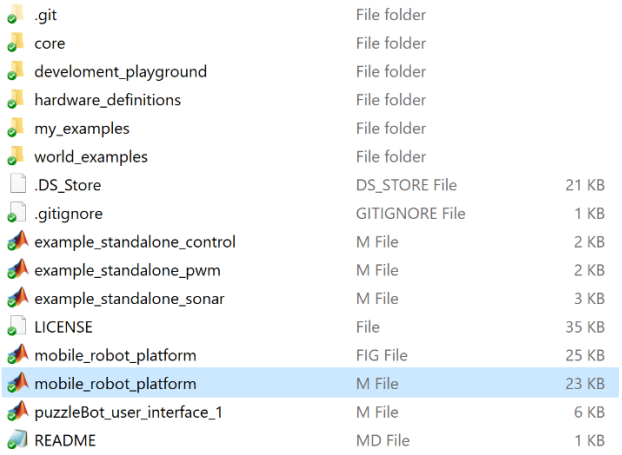


FIGURE 57 MRUI FOLDER FILES

3. Open MATLAB and change the current folder address tab in MATLAB to the path where the MRUI folder is located; you can verify this since the folders inside MRUI appear in the current folder tab.

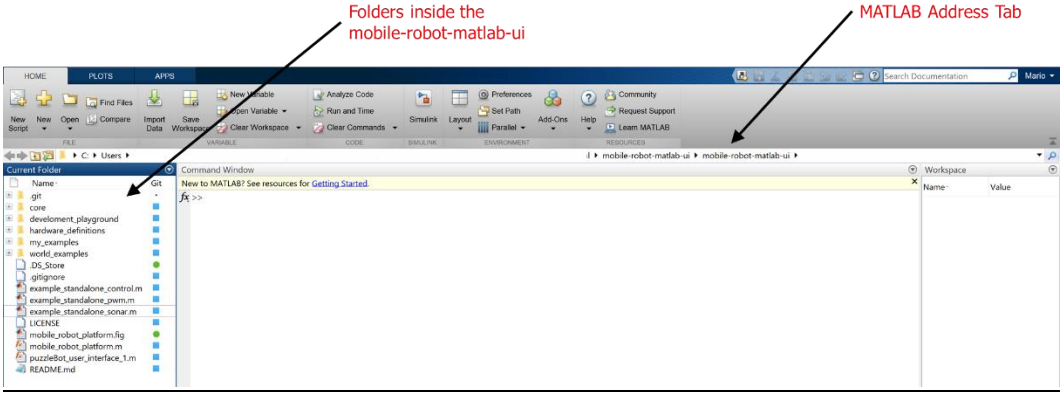


FIGURE 58 MATLAB SETUP:SETTING PATH

4. Create a New Script from the Home Tab>New Script as shown in the following figure.

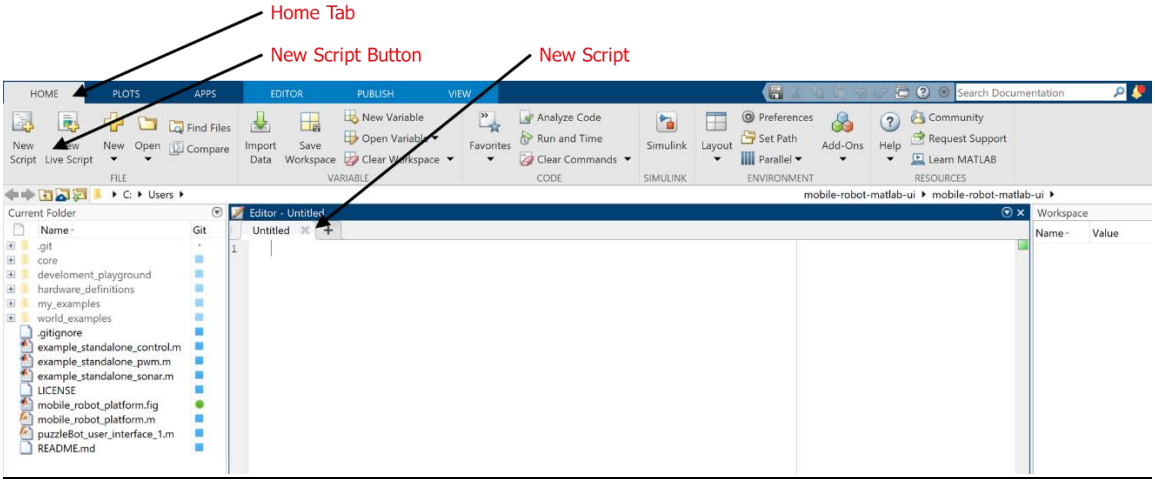


FIGURE 59 MATLAB NEW SCRIPT

5. Now the user can start using the libraries provided by Manchester Robotics ltd. to control the real/simulated robot. More information about these libraries as well as some examples can be found in the subsequent sections.
6. Once the Program is done, it can be run by going into the Editor Tab>Run

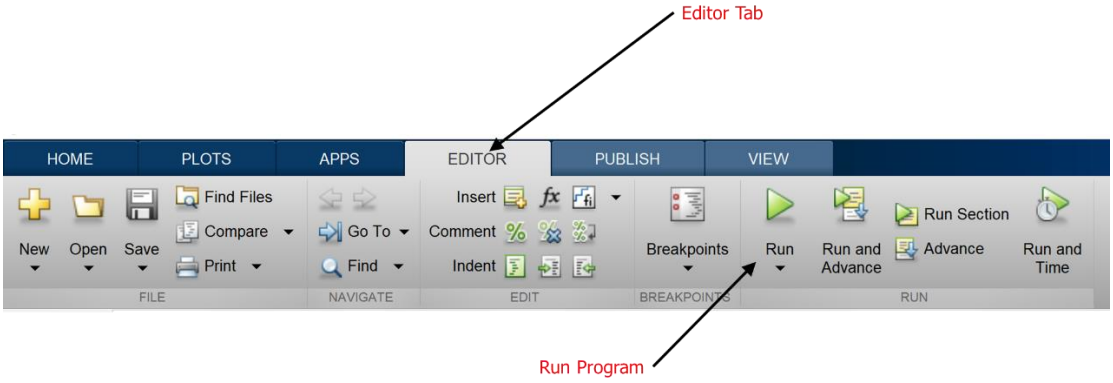


FIGURE 60 RUNNING THE PROGRAM

EXAMPLES STAND-ALONE STYLE (SA-MRUI)

The following examples are for the Stand-Alone style, this require the Real robot to receive PWM signals to the motor as seen in Configuration File (YAML file) section when defining the Main Parameters of the Robot>Robot>Control Input>3 (for PWM control input).

The following examples follow the basic workflow shown in Figure 55.

EXAMPLE STAND-ALONE PWM

This example simulates a robot (can be connected to the real robot), into a basic world and sends some PWM signals to the motors, saves the data and plots the speeds of the wheels given by the encoders.

- This example Simulates a robot (robot_0002 can be found in the folder hardware_definitions>robot_0002) comprised of a differential drive robot (DDR)with two motors with encoders in each motor and a Sonar attached to a Servo motor.
- Detailed comments are provided in the example file. In order to use the real robot uncomment the line and set the robot IP as follows *R1.connect('ROBOT IP')*.
- The simulated world used for this test is the world_0002 consisting in world with basic figures (world_examples>world_0002.json).

SEND THE PWM SIGNALS [-1, 1] AS DEFINED IN

- General Specs, directly to the Robot Motors.
- Using the PWM signals the robot is updated via *R1.update()* and now the user can access the sensor information using the appropriate topic.
- Save the data provided by the encoders in each motor and when the simulation finishes, plots the results.
- To run the following example, follow step 5 and 6 in Setting Up The MATLAB Control Environment.

```
clc, clear all
restoredefaultpath
addpath(genpath(pwd))


% Create Robot
R1 = RobotClass('json_fname', 'robot_0002.json');
% Uncomment below for using the real robot (it's simulation otherwise)
% R1.connect('192.168.1.1');

% Create World
W = WorldClass('fname', 'world_0002.json');

% Total duration and sampling time parameters
TotalTime = 1;
t_sampling = 0.02;

t_start = tic;
t_loop = tic;

% Wheel control pwm signals [-1,1]
uR = 1;
uL = 0.5;

wR_all = [0];
wL_all = [0];

while toc(t_start)<TotalTime

    dt = toc(t_loop);

    if(dt>=t_sampling) % execute code when desired sampling time is reached
        t_loop = tic;

        % Update robot in this order: actuators, pose (in simulation), sensors
        if toc(t_start)>0.0
        actuator_signals = {'right motor', uR, 'left motor', uL};
        else
        actuator_signals = {'right motor', 0, 'left motor', 0};
        end
        sensor_readings = R1.update(dt, W, 'kinematics', 'voltage_pwm', actuator_signals{:});

        % Update encoder velocity readings
        wR = sensor_readings('right encoder');
        wL = sensor_readings('left encoder');

        % Save data for ploting
        wR_all = [wR_all wR];
        wL_all = [wL_all wL];
    end

     pause(0.001)
end

% plot saved velocities for right and left wheels
plot(wR_all);
hold on
```

```
plot(wL_all);
```

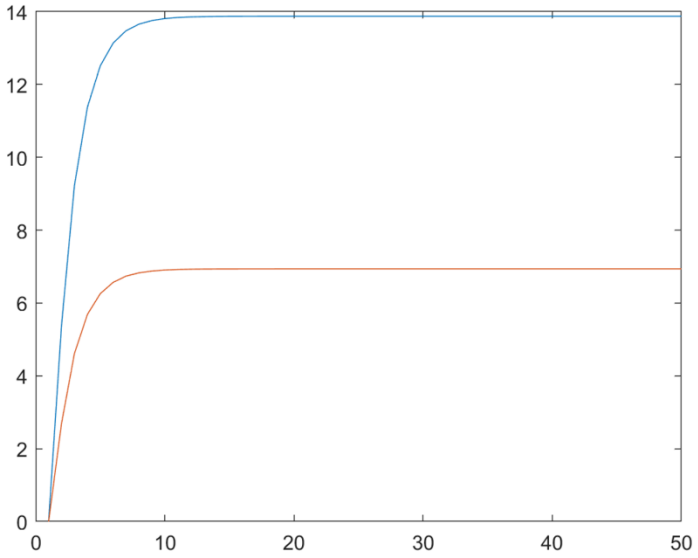
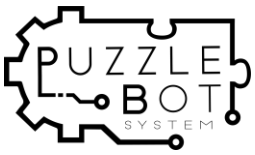


FIGURE 61 OUTPUT OF THE STAND-ALONE PWM EXAMPLE



EXAMPLE STAND-ALONE CONTROL

This example simulates a robot (can be connected to the real robot), into a basic world. A PID controller is then used to provide the PWM signals to the motors in order to achieve the set points, saves the data coming from encoders and plots the speeds of the wheels. In this example a PID controller library provided by Manchester Robotics ltd. is used, the user can access this library in core>utils>MotorControl.m.

```
clc, clear all
restoredefaultpath
addpath(genpath(pwd))


% Create Robot
R1 = RobotClass('json_fname', 'robot_0002.json');
% Uncomment below for using the real robot (it's simulation otherwise)
% R1.connect('192.168.1.1');

% Create World
W = WorldClass('fname', 'world_0002.json');

% Total duration and sampling time parameters
TotalTime = 2;
t_sampling = 0.02;

t_start = tic;
t_loop = tic;

% Initialise motor angular velocity controllers
control_right = MotorControl();
control_left = MotorControl();

% Controller setpoints for right and left wheel
wR_set = 7;
wL_set = 6;

wR = 0;
wL = 0;

wR_all = [0];
wL_all = [0];

while toc(t_start)<TotalTime

    dt = toc(t_loop);

    if(dt>=t_sampling)              % execute code when desired sampling time is reached
        t_loop = tic;

        % Right wheel controller %%%%%%%%%%%%%%%%%%%%
        errR = wR_set - wR;

        uR = control_right.Control(errR,dt);
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        % Left wheel controller %%%%%%%%%%%%%%%%%%%%
        errL = wL_set - wL;

        uL = control_left.Control(errL,dt);
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        % Update robot in this order: actuators, pose (in simulation), sensors
        actuator_signals = {'right motor', uR, 'left motor', uL};
        sensor_readings = R1.update(dt, W, 'kinematics', 'voltage_pwm', actuator_signals{:});

        % Update encoder velocity readings
```

```
        wR = sensor_readings('right encoder');
        wL = sensor_readings('left encoder');

        % Save data for ploting
        wR_all = [wR_all wR];
        wL_all = [wL_all wL];

    end

     pause(0.001)
end

% Plot wheel angular velocities
plot(wR_all);
hold on;
plot(wL_all);
```

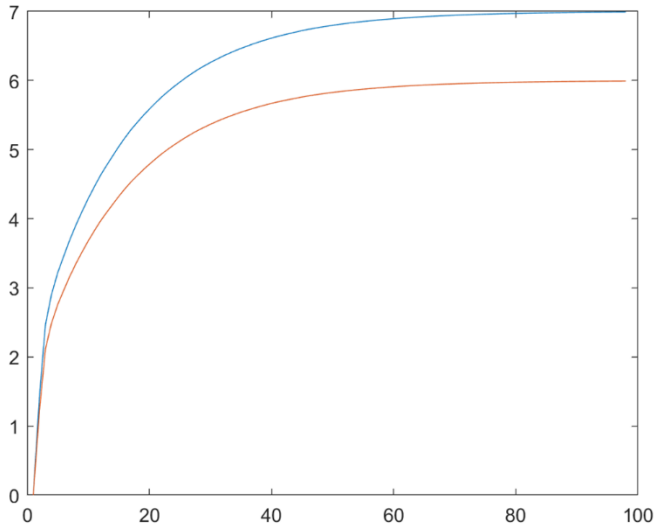


FIGURE 62 OUTPUT FROM STAND-ALONE CONTROL EXAMPLE

EXAMPLE STAND-ALONE SONAR

This example simulates a robot (can be connected to the real robot), into a world with a wall in the shape of a circle as shown in Figure 63. A cascade control strategy is used keep the robot at a desired distance from the obstacle. In this example the outer loop is in shcarge with keeping the constant distance to the obstacle and provides the set-points to the inner loop. The inner loop then runs a PID controller that provides the PWM signals to the motors. The program plots the position of the robot iteratively to simulate the movement of the robot.

```
clc, clear all, close all
restoredefaultpath
addpath(genpath(pwd))


% Create Robot (robot with reflectance,sonar and servo motor)
R1 = RobotClass('json_fname', 'puzzle_bot_0002.json');
% Uncomment below for using the real robot (it's simulation otherwise)
% R1.connect('192.168.1.1');

% Create World
W = WorldClass('fname', 'world_wall.json');

% Plot world and robot
figure(1)
h_w = W.plot();

% Total duration and sampling time parameters
TotalTime = 10;
sampling_inner = 0.02;
sampling_outer = 0.1;

t_start = tic;
t_outer_loop = tic;
t_inner_loop = tic;

% Initialise motor angular velocity controllers
control_right = MotorControl();
control_left = MotorControl();

% Desired wheel velocities
wR_set = 0;
wL_set = 0;

wR = 0;
wL = 0;

% desired wheel velocity
w_desired = 5;

% Proportional gain for following the wall
K = 5;

while toc(t_start)<TotalTime

    %% Outer loop
    dt = toc(t_outer_loop);

    if(dt>=sampling_outer) % execute code when desired outer loop sampling time is reached
        t_outer_loop = tic;

        sonar_dist = sensor_readings('sonar');

        % If a wall is found less than 1m away from the robot then follow it
        if sonar_dist < 1
            err_dist = 0.5 - sonar_dist;  % hold a constant distance of 0.5m to the wall
            wR_set = w_desired - K*err_dist;
```

```
            wL_set = w_desired + K*err_dist;
        else
            wR_set = 0;
            wL_set = 0;
        end

        % Plot all
        try
            delete([h_r, h_s])
        catch
        end
        h_r = R1.plot('simple');
        h_s = R1.plot_measurements('sonar');
    end

    %% Inner loop
    dt = toc(t_inner_loop);
    if(dt>=sampling_inner) % execute code when desired inner loop sampling time is reached
        t_inner_loop = tic;

        uR = control_right.Control(wR_set-wR,dt);
        uL = control_left.Control(wL_set-wL,dt);

        % Update robot in this order: actuators, pose (in simulation), sensors
        % Set the servo angle to 60 degrees to the left, 1 radian (where the wall should be)
        actuator_signals = {'right motor', uR, 'left motor', uL,'servo motor',1};
        sensor_readings = R1.update(dt, W, 'kinematics', 'voltage_pwm', actuator_signals{:});

        % Update encoder velocity readings
        wR = sensor_readings('right encoder');
        wL = sensor_readings('left encoder');

    end

    pause(0.001)
end

actuator_signals = {'right motor', 0, 'left motor', 0,'servo motor',0};
R1.update(dt, W, 'kinematics', 'voltage_pwm', actuator_signals{:});
```

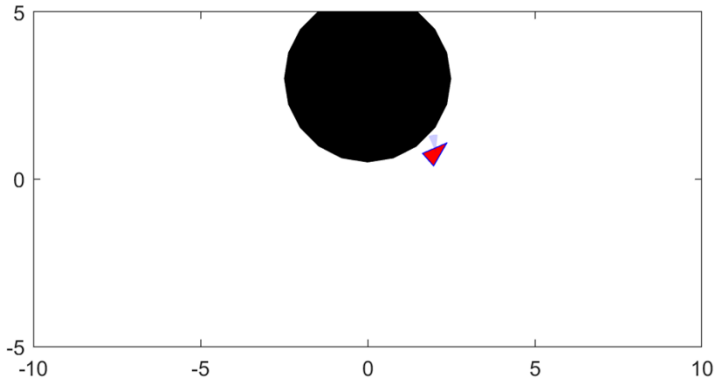


FIGURE 63 OUTPUT OF THE STAND-ALONE SONAR EXAMPLE

ARDUINO STYLE (A-MRUI)

This is another style of programming the MATLAB Robotic User Interface, different from the Stand-Alone Version, where the user had to develop from scratch the control loop, select when to update the simulated robot states and develop its own GUI, using the libraries provided by Manchester Robotics ltd.

In this style of programming the MATLAB Robotic User Interface runs a predefined Continuous Loop, automatically updating the information from the robot sensors and actuators, the user control code, and the Graphical User Interface (GUI).

In other words, the main difference between this programming style and the one mentioned in the Stand Alone Style section, is that in this style the user does not have to worry about updating the robot, setting the control loop or implementing a graphic user interface, but only on the control algorithm for the robot.

As stated before, in this type of programming style, the user code runs repeatedly inside the loop as shown in the following figure.

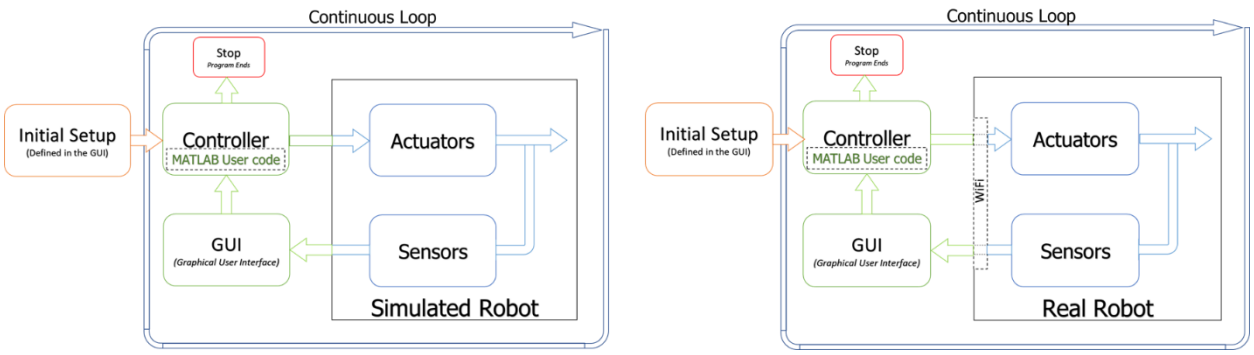


FIGURE 64 ARDUINO STYLE PROGRAMMING

To make the programming of the Controller easy for the user, the MATLAB User Code section is subdivided into two sections based on the Arduino programming paradigm. *The setup section* and the *loop section*.

The *setup section* consists of a part of the program used for setting up the variables to be used by the program, this section runs only one time in the continuous loop.

The *loop section* is the section that run continuously with the loop. This section defined by the user will oversees the updating the signals to be sent or manage the ones received from to/from the actuators and/or sensors, as well as implementing the control algorithm defined by the user. The loop and setup section can be seen the following diagram.

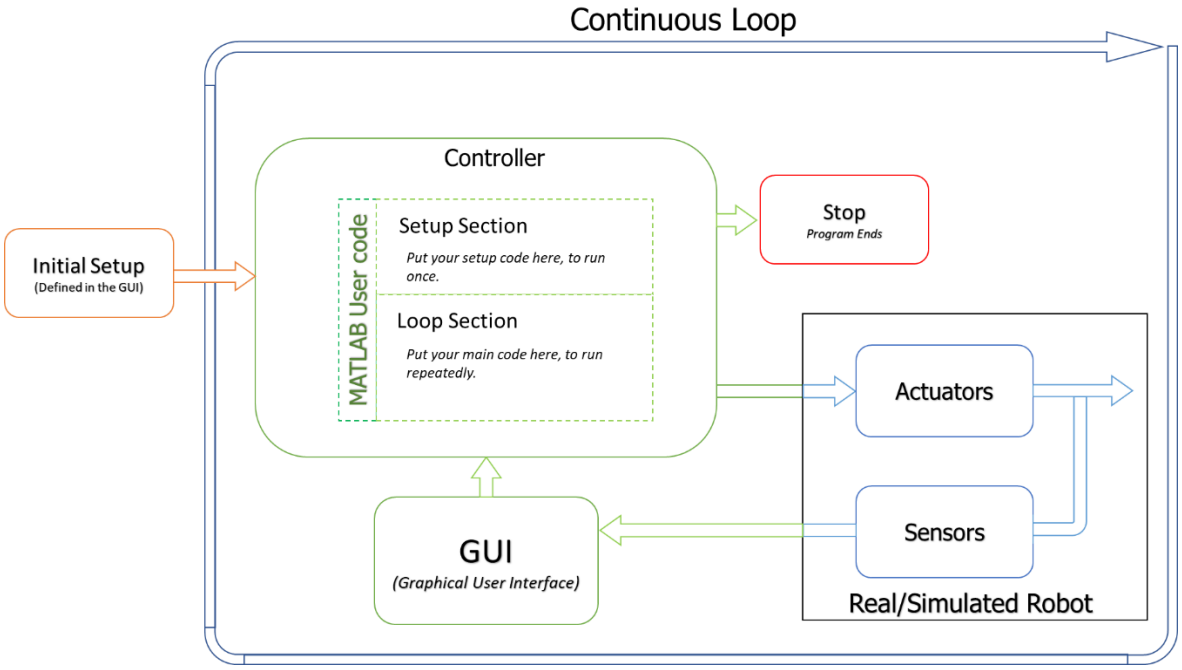


FIGURE 65 SETUP AND LOOP SECTIONS IN THE MATLAB USER CODE

In the following sections some basic functionalities and examples on how to use the MATLAB Robotic User Interface with this programming style will be shown.

Note, the Continuous Loop (Big Loop) described in the figure by the blue arrow surrounding everything; can also be configured by the user for further reference go to the Advance user Manual.

STARTING THE MATLAB ROBOTIC USER INTERFACE (MRUI)

In this section a basic tutorial on how to set up the *MATLAB Robotic User Interface* by *Manchester Robotics ltd.* will be shown.

1. Unzip the MATLAB Robotic User Interface (*mobile-robot-matlab-ui.rar*)

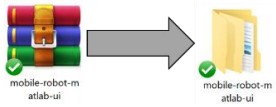


FIGURE 66 UNZIP THE MATLAB ROBOTICS USER INTERFACE

2. The MRUI folder contains different files as shown in the following figure, each file will be explained in the subsequent sections.

.git	File folder	
core	File folder	
develoment_playground	File folder	
hardware_definitions	File folder	
my_examples	File folder	
world_examples	File folder	
.DS_Store	DS_STORE File	21 KB
.gitignore	GITIGNORE File	1 KB
example_standalone_control	M File	2 KB
example_standalone_pwm	M File	2 KB
example_standalone_sonar	M File	3 KB
LICENSE	File	35 KB
mobile_robot_platform	FIG File	25 KB
mobile_robot_platform	M File	23 KB
puzzleBot_user_interface_1	M File	6 KB
README	MD File	1 KB

FIGURE 67 MRUI FOLDER FILES

3. Open the MRUI by selecting the *mobile_robot_platform.m* file. Shown in Figure 67 in blue.
4. The following screen should appear

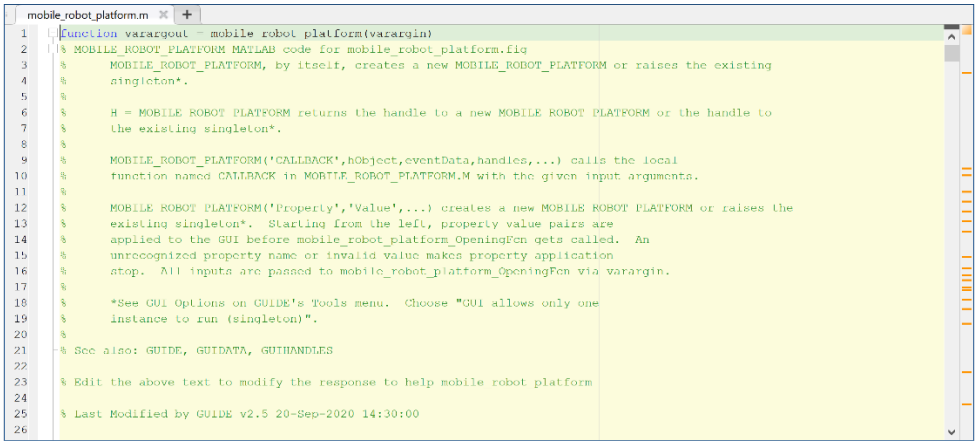


FIGURE 68 MATLAB MRUI

5. Make sure the current folder address tab in MATLAB has the folder address where the MRUI folder is located; by verifying the folders inside MRUI appear in the current folder tab. Note If you click the Run button, the directory tab changes automatically, but sometimes this might fail due to MATLAB.

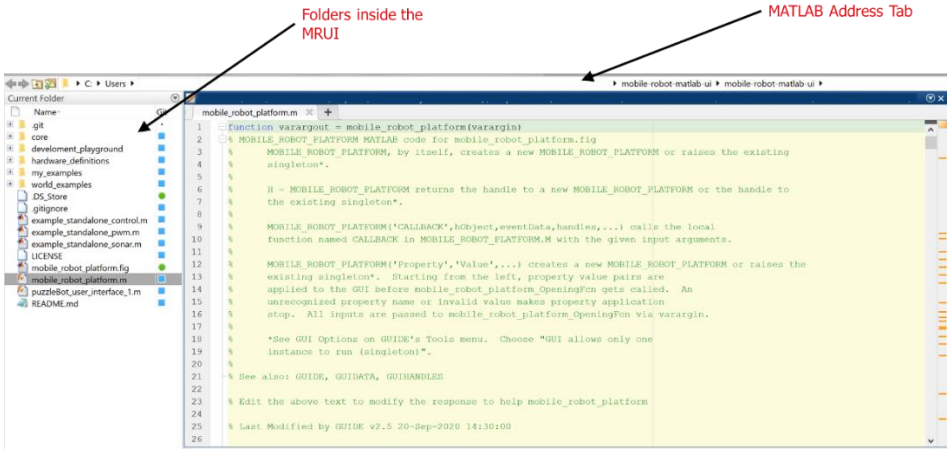


FIGURE 69 FOLDERS IN MRUI

6. Run the program from the editor tab in MATLAB.

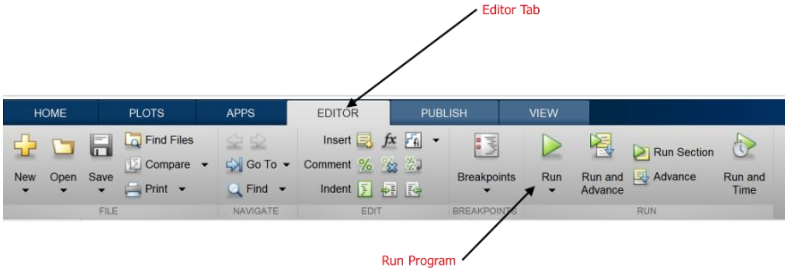


FIGURE 70 EDITOR TAB AND RUN BUTTON IN MATLAB

7. The following screen should appear. *Note, since there is no control program at his time the Robot (represented by the red triangle will not move). Follow the steps in the next section to run an example program.*

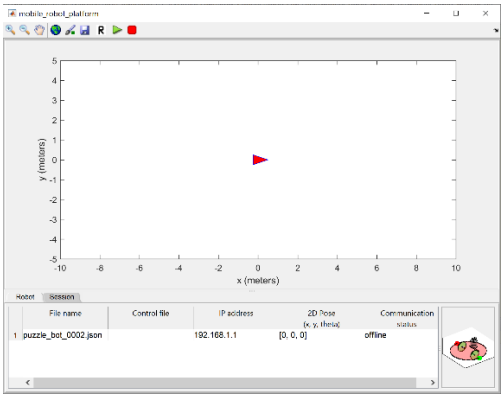


FIGURE 71 MOBILE ROBOTIC USER INTERFACE

RUNNING A PROGRAM IN THE MATLAB ROBOTIC USER INTERFACE

For this section the example program *drive_straight* located in *mobile-robot-matlab-ui/my_examples* will be used to show the user how to tun a program in the GUI.

1. Follow the steps in section Starting the MATLAB Robotic User Interface (MRUI).
2. Select the Robot to be used for this simulation. For this example, the robot described by the JSON file *puzzle_bot_0002.json (default)* will be used. This robot is comprised by a Differential drive robot with reflectance, servo and a sonar attached to the servo.

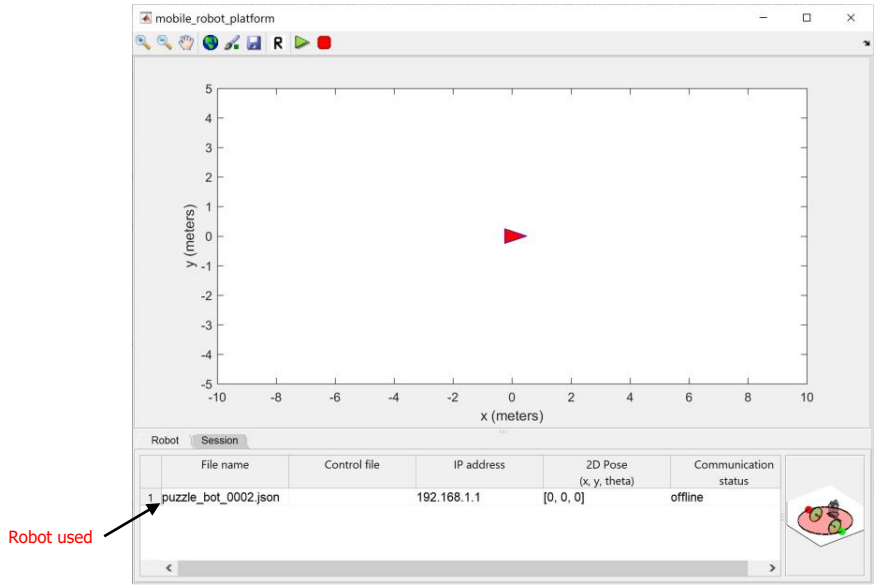


FIGURE 72 SET UP THE ROBOT FOR SIMULATION

3. Set Up the Control File used for this simulation. For this case, the example file *drive_straight* will be used.

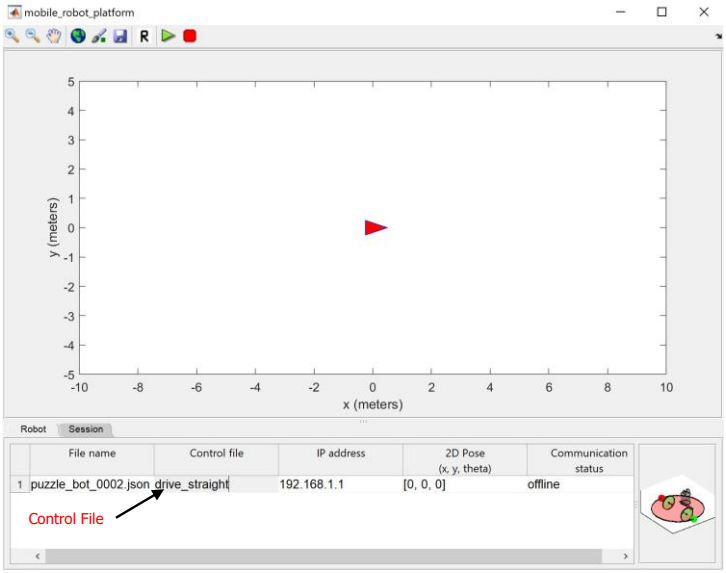


FIGURE 73 SET UP THE CONTROL FILE

4. For this case since it is a simulation the IP Address and the Communication status of the robot can be left 'as is'.

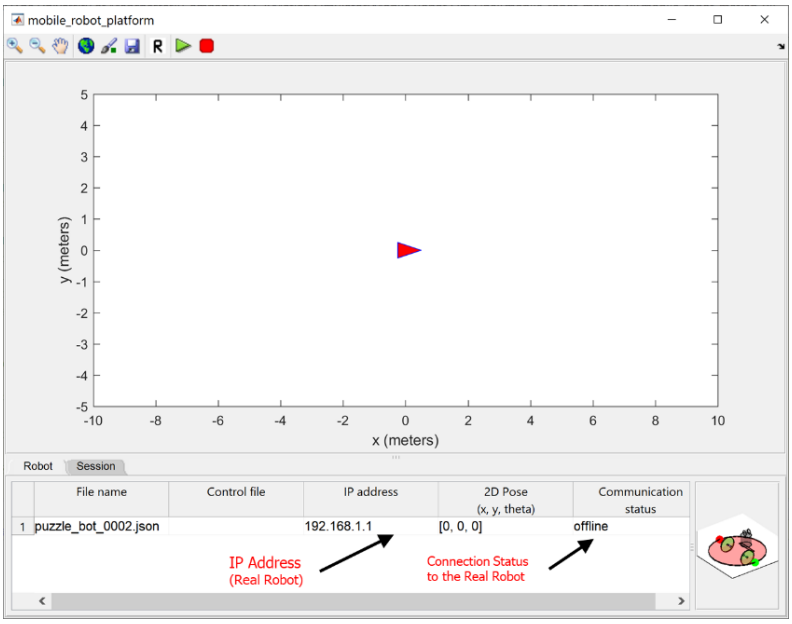


FIGURE 74 IP ADDRESS AND COMMUNICATION STATUS TAB

5. The user can change the initial position of the robot by changing the values in the pose tab.

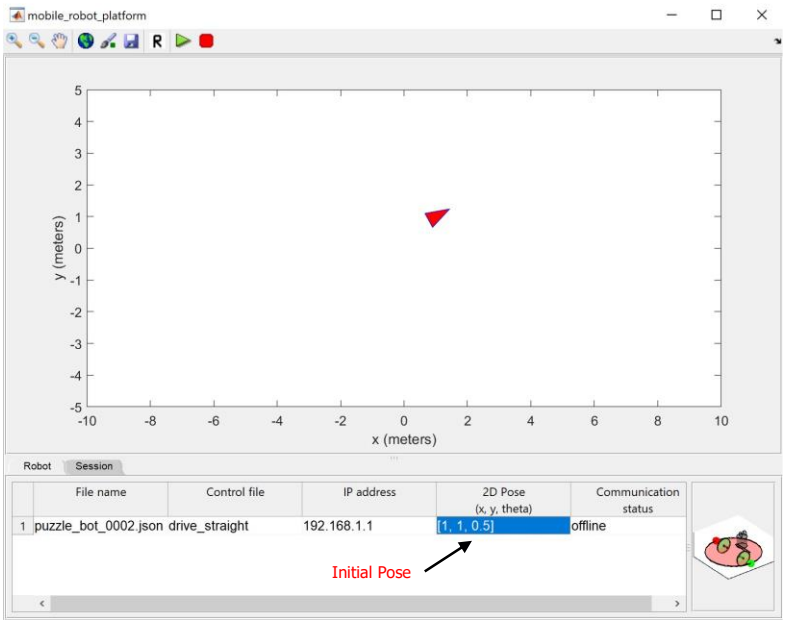


FIGURE 75 SET UP THE ROBOT POSITION

6. The World used for the simulation can be defined by clicking the World Button and selecting it from a JSON file. For this example, the default empty world will be used.

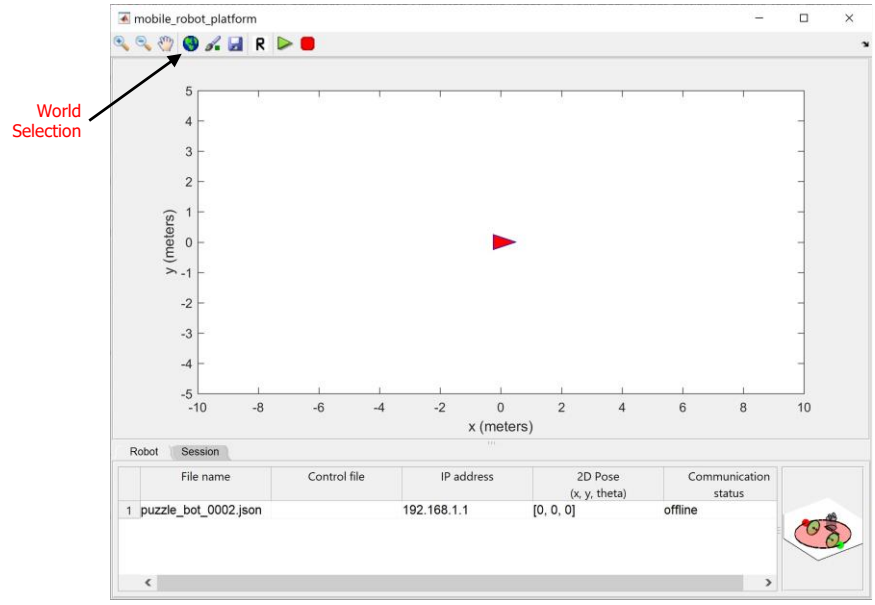


FIGURE 76 SET UP THE WORLD TO BE USED

7. Use the *Run* and *Stop* buttons to start and stop the simulation and the Pan and Zoom buttons to move inside the plot.

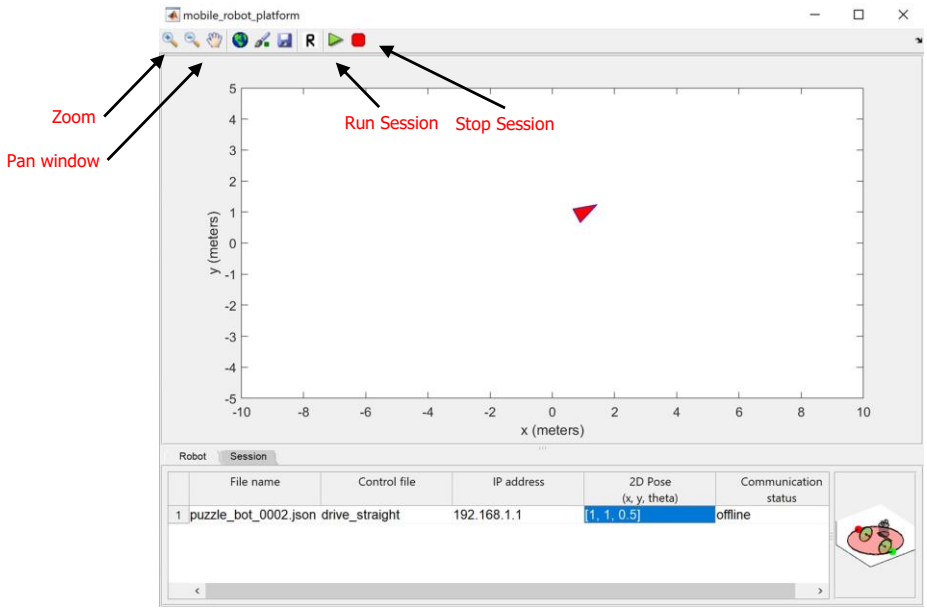


FIGURE 77 START AND STOP THE SIMULATION

8. The result should look like the following image. Note the rest of the GUI function will be described in the subsequent sections.

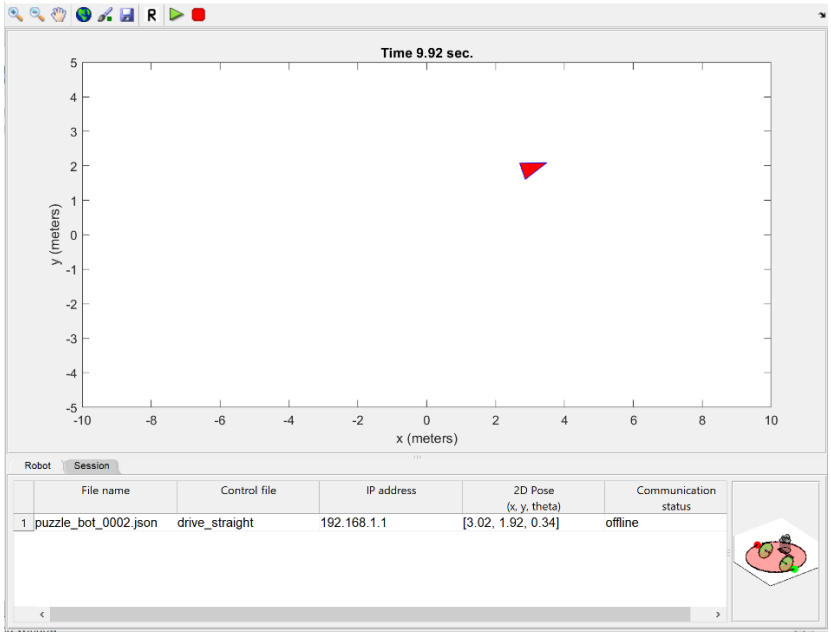


FIGURE 78 ROBOT AFTER SIMULATION

CONNECT TO REAL ROBOT

In this section, the steps of connecting the real robot to the GUI and testing the code with the real robot will be shown.

1. Check the IP address in the GUI and the robot IP address that is shown in the screen of the real robot. If they are different, change the IP address in the GUI to be the same as the one shown on the robot screen.

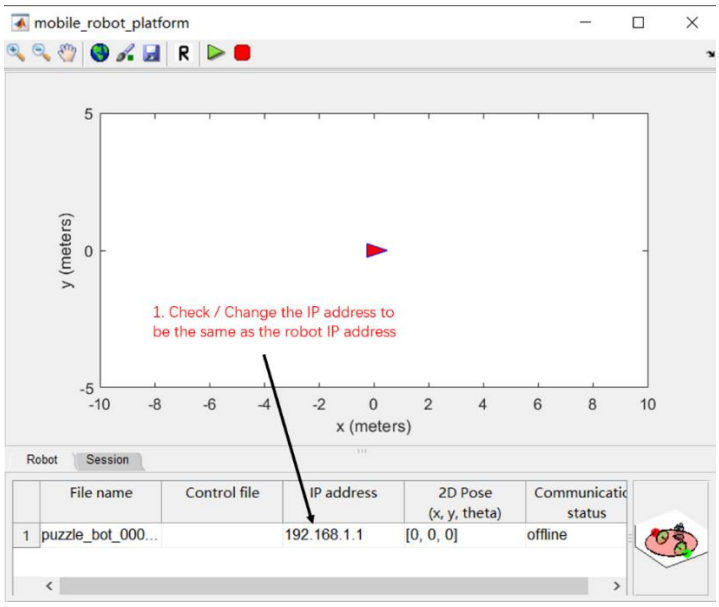


FIGURE 79 ROBOT IP ADDRESS

2. Select the 'Session' tab, and then tick the checkbox 'Connect to physical robot'

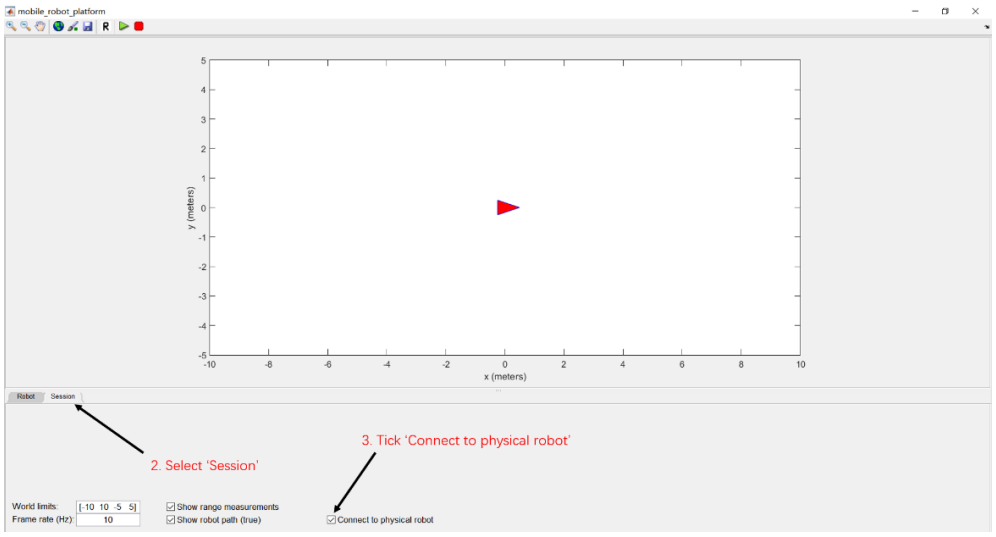
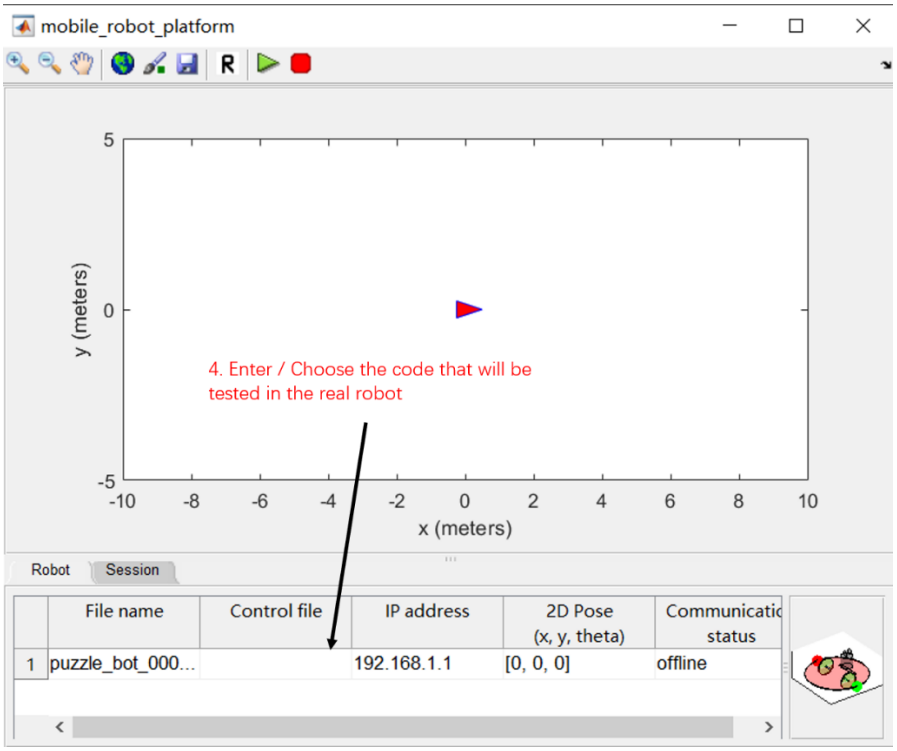


FIGURE 80 REAL ROBOT CONNECTION

3. Click the tap ‘Robot’ tab and enter the code that will be tested on the real robot and run the program.



GUI DESCRIPTION

As stated in the previous sections, the GUI sets some basic variables used in the program, such as the type of robot, the world, the control file and the connection to the robot. At the same time it works as a user interface with the robot for the user. In this section rest of the functions from the GUI will be described.

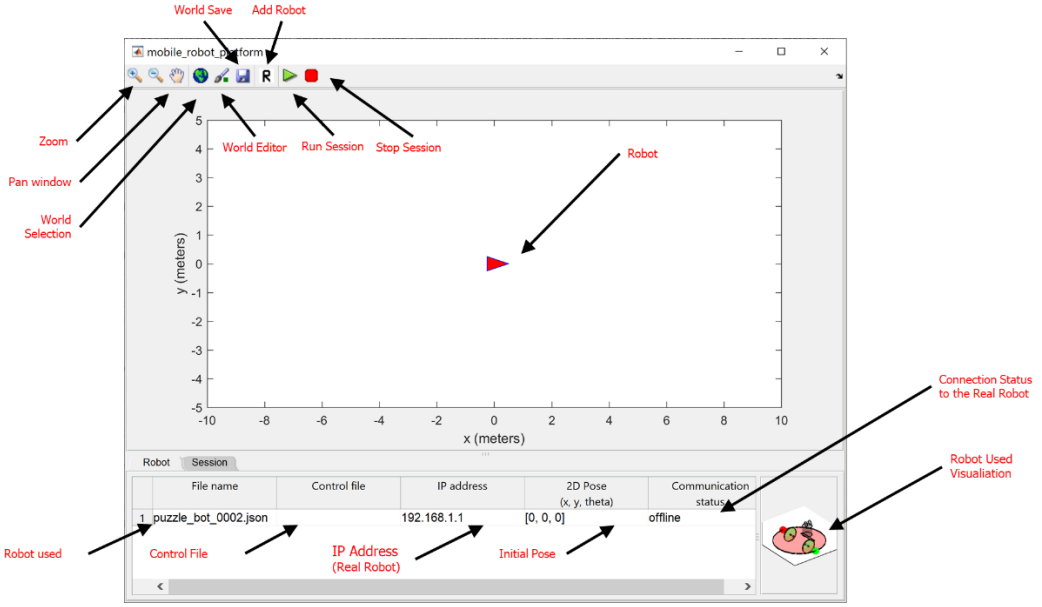


FIGURE 81 MATLAB ROBOTIC USER INTERFACE GUI

TABLE 9 GUI DESCRIPTION TABLE

GUI	Description
Robot	Robot representation in the GUI.
Pan, Zoom	Pan and Zoom in the plot area.
World Selection	Select the JSON file containing the world. File selector window will pop-up.
World Editor	Edit the current World. See subsequent sections.
World Save	Save the edited world.
Add Robot	Add a new robot. For multiple robot simulation (Advanced users).
Run Session	Start simulation/ real robot.
Stop Session	Stop the session
File Name	Set the JSON file defining the robot to be used.
Control File	Set the control file to be used.
IP Address	IP address of the real robot.
2D Pose	Initial Pose of the robot.
Connection Status	Status Connection of the real robot.
Robot Visualisation	Visualisation of the Robot Used (defined by the JSON file in the File Used tab)

WORLD EDIT

One of the main characteristics of the GUI is the flexibility given to the user to edit the current world, generate its own world and save it for future use. In this section a basic world will be done as an example, using the empty (default world).

1. Initialise the MATLAB Robotic User Interface as done in the section: Starting the MATLAB Robotic User Interface (MRUI).
2. Click on the World Editor Button.

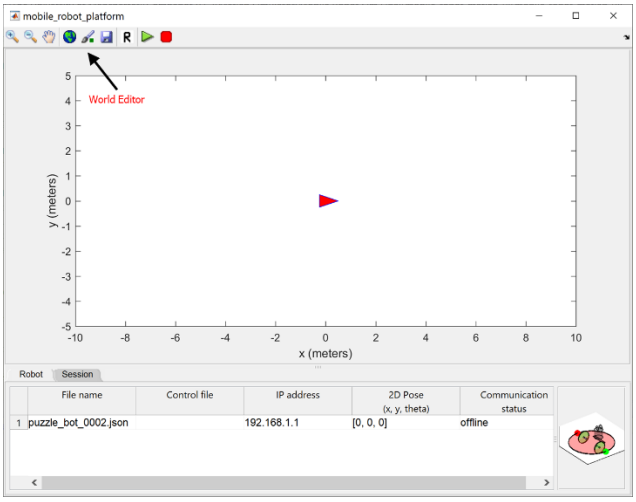


FIGURE 82 WORLD EDITOR BUTTON

3. The following World Editor Mode Screen will appear.

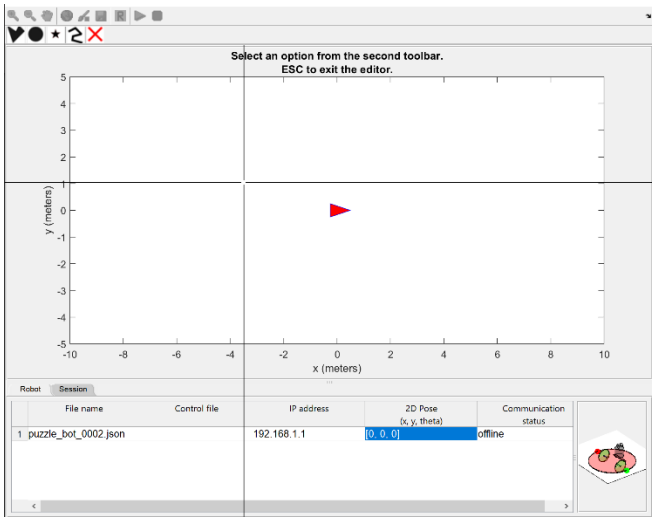


FIGURE 83 WORLD EDITOR MODE

4. Follow the instructions given in the top bar of the GUI. To Exit the Editor Mode press ESC on the keyboard.

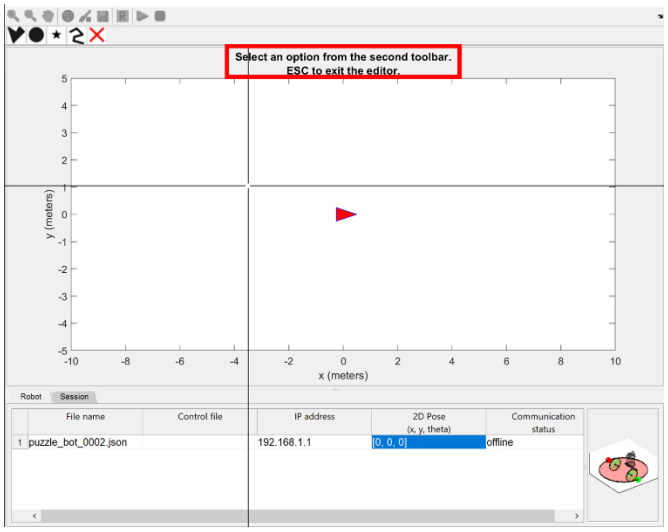


FIGURE 84 INSTRUCTION FOR EDITING THE WORLD.

The user can define different objects in the map such as Polygons, Circles, Points and a Path (for line following). For each object follow the instructions at the top bar (Figure 84).

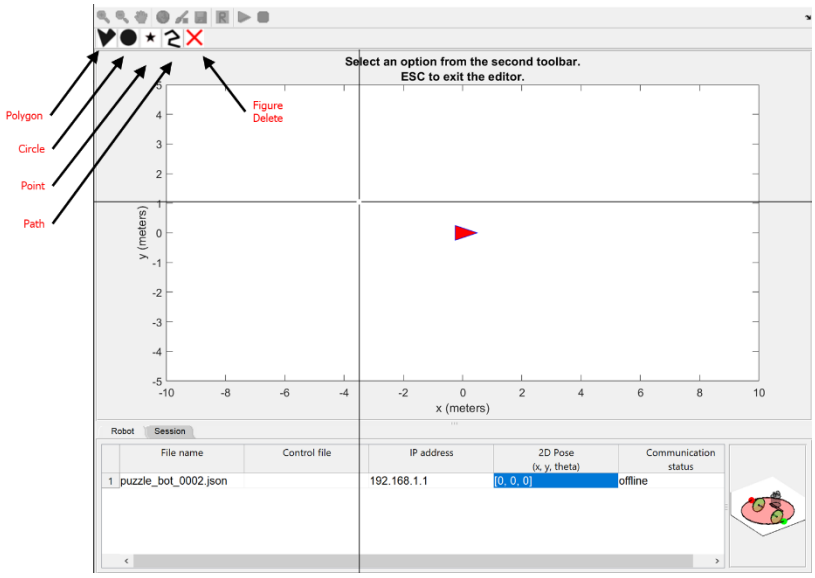


FIGURE 85 OBJECT SELECTOR.

5. For the Polygon shape use the mouse to move the Crosshair and define the points of the obstacle and the SPACE key (in the keyboard) to close the shape.
6. For the circle use the crosshair to define the centre and click again to define the radius.
7. For the point just select using the crosshair the position.
8. For the path use the crosshair to define the points of the path.

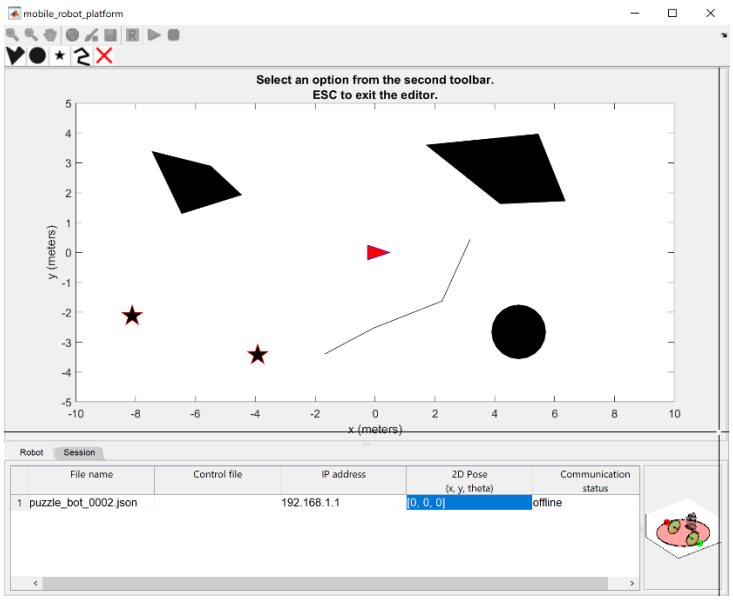


FIGURE 86 DIFFERENT FIGURES DRAWN USING THE WORLD EDITOR.

9. To delete a figure, select the Figure Delete button and use the crosshair to delete the selected figure.
10. Once you have finished editing the World, press ESC to go back to normal mode.
11. Save the World for future use by pressing the Button Save World. A pop-up window will appear, name your world and then press Save.

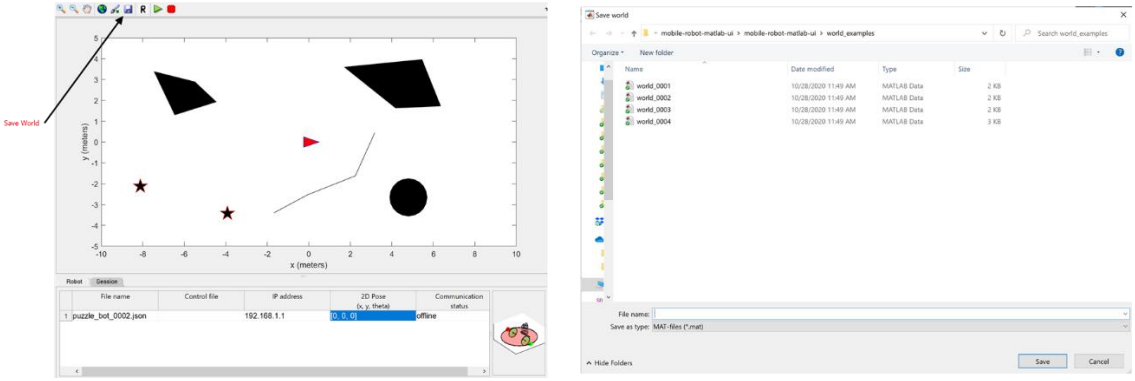


FIGURE 87 SAVE WORLD

EXAMPLE OF A BASIC CONTROL ALGORITHM

In this section a basic control algorithm will be presented. This example can be found in *my_examples>drive_straight*.

Recall the way the MATLAB Robotic User Interface program works in the following diagram.

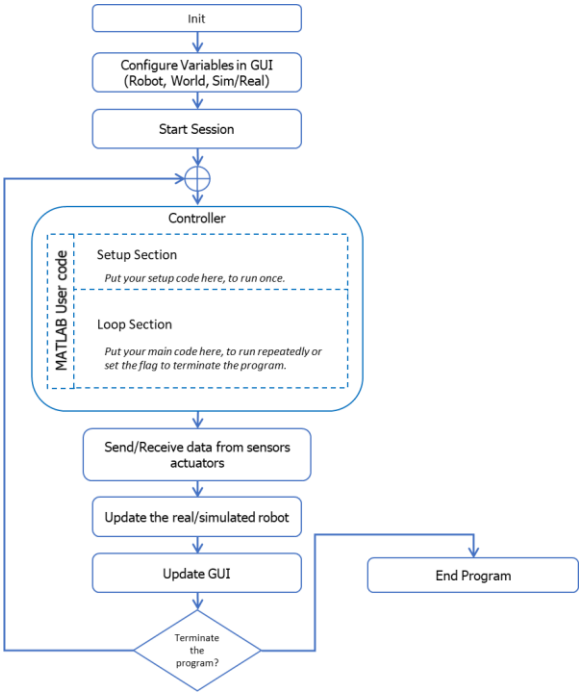


FIGURE 88 BASIC WORKFLOW OF THE MRUI FOR THE ARDUINO PROGRAMMING STYLE

As stated previously the controller part programming defined by the user is based on the Arduino style of programming. In which the user has a *setup section* and a *loop section* to run its code one time and multiple times, respectively.

```
%% MATLAB User Code Empty Template
function my_alg = name(my_alg, robot)
%% Variable Definition


%% Setup Section
if my_alg('is_first_time')
end

%% Loop Section


return
```

```
sketch_nov01a
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

FIGURE 89 CONTROL TEMPLATE COMPARISON WITH ARDUINO TEMPLATE

As seen in the previous figure, the control function template contains two pre-defined structures *my_alg* and *robot*.

The *robot* container structure handles all the internal variables of the robot for the proper function of the MATLAB Robotics User Interface. Modifying this structure is not recommended for basic users.

The *my_alg* is a container (hash table of the class container.map) dedicated for the user to define and add its own variables or problem-specific parameters as elements to the my_alg structure that will persist throughout the session. At the same time this container has some global elements to be understood by the MATLAB Robotics User Interface that contain information regarding simulation variables to be managed by the user. The following table shows the global elementes pre-defined in *my_alg*.

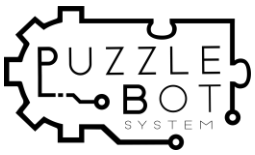
TABLE 10 MY_ALG STRUCTURE

MyAlg Structure	Description
my_alg('tic')	Returns an identifier to the start time of session. To compute the current time use toc(my_alg('tic'))
my_alg('is_first_time')	Returns a boolean to indicate whether the iteration is the first one. This element can be used to initialize parameters needed for the algorithm.
my_alg('is_done')	Set a boolean to terminate the session programmatically.
my_alg('dc_motor_signal_mode')	Returns a string defining the signal mode. Options are {'omega_setpoint', 'voltage_pwm'}
my_alg(...)	Returns the value of the actuator or sensor labelled ... in robot JSON-file as follows.
• my_alg('right encoder')	Right encoder velocity
• my_alg('left encoder')	Left encoder velocity
• my_alg('reflectance')	Reflectance sensor output value
• my_alg('sonar')	Sonar measured distance (m)
• my_alg('right motor')	Sets the right motor input signal (pwm or angular velocity)
• my_alg('left motor')	Sets the left motor input signal (pwm or angular velocity)
• my_alg('servo motor')	Sets the servomotor angle (radians)
my_alg('Name')	Definition of a variable to be added to *my_alg* by the user. The 'Name' can be replaced by any name defined by the user.

EXAMPLE BASIC

```
function my_alg = drive_straight(my_alg, robot)
% This function drives the robot in a straight line for 10 seconds.
% Then, it stops.
%
% Mohamed Mustafa, August 2020
% ------------------------------------------------------------------------

if my_alg('is_first_time')
    my_alg('dc_motor_signal_mode') = 'omega_setpoint';     % change if necessary to 'voltage_pwm'
end

time = toc(my_alg('tic'));      % Get time since start of session
if time < 10
    % Drive
    my_alg('right motor') = 5;
    my_alg('left motor') = 5;
else
    % Stop motors
    my_alg('right motor') = 0;
    my_alg('left motor') = 0;
    % Stop session
    my_alg('is_done') = true;
end

% display encoder readings
left_encoder_omega = my_alg('left encoder');
right_encoder_omega = my_alg('right encoder');
%['Encoders (left, right): (' num2str(left_encoder_omega) ', ' num2str(right_encoder_omega) ')
rad/sec.']

return
```

To run this example, follow the steps defined in section: Running a Program in the MATLAB Robotic User Interface.



OTHER CONTROL ALGORITHM EXAMPLES

```
function my_alg = example_control(my_alg, robot)
% This function implements velocity controllers for both wheels
% and applies the desired setpoints for a specified amount of time.
%
% Mohamed Mustafa, August 2020
% -------------------------------------------------------------------------
%
% Reading data from sensors (if present on the robot)
%    my_alg('right encoder') - right encoder velocity
%    my_alg('left encoder')  - left encoder velocity
%    my_alg('reflectance')   - reflectance sensor output value
%    my_alg('sonar')         - sonar measured distance (m)
%
% Sending controls to actuators (if present on the robot)
%    my_alg('right motor')   - sets the right motor input signal (pwm or angular velocity)
%    my_alg('left motor')    - sets the left motor input signal (pwm or angular velocity)
%    my_alg('servo motor')   - sets the servomotor angle (radians)
% -------------------------------------------------------------------------

if my_alg('is_first_time')
    %% Setup initial parameters here

    my_alg('dc_motor_signal_mode') = 'voltage_pwm';     % change if necessary to 'omega_setpoint'

    % Initialise wheel angular velocity contollers
    my_alg('wR_set') = 7;
    my_alg('wL_set') = 6;

    my_alg('control_right') = MotorControl();
    my_alg('control_left') = MotorControl();

    % Initialise vectors for saving velocity data
    my_alg('wR_all') = [];
    my_alg('wL_all') = [];

    % Initialise time parameters
    my_alg('t_sampling') = 0.02;
    my_alg('t_loop') = tic;
    my_alg('t_finish') = 2;
end

%% Loop code runs here

time = toc(my_alg('tic'));      % Get time since start of session

if time < my_alg('t_finish')    % Check for algorithm finish time

    dt = toc(my_alg('t_loop'));

    if dt>my_alg('t_sampling')  %
        my_alg('t_loop') = tic;

        % Right wheel controller %%%%%%%%%%%%%%%%%%%%
        errR = my_alg('wR_set') - my_alg('right encoder');

        uR = my_alg('control_right').Control(errR,dt);
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        % Left wheel controller %%%%%%%%%%%%%%%%%%%%
        errL = my_alg('wL_set') - my_alg('left encoder');

        uL = my_alg('control_left').Control(errL,dt);
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        % Apply pwm signal
        my_alg('right motor') = uR;
        my_alg('left motor') = uL;
```

```
        my_alg('wR_all') = [my_alg('wR_all') my_alg('right encoder')];
        my_alg('wL_all') = [my_alg('wL_all') my_alg('left encoder')];

    end

else
    %% Finish algorithm and plot results

    % Stop motors
    my_alg('right motor') = 0;
    my_alg('left motor') = 0;
    % Stop session
    my_alg('is_done') = true;

    % Plot saved velocities for right and left wheel
    figure(2);
    plot(my_alg('wR_all'));
    hold on
    plot(my_alg('wL_all'));

end

return
```

Note Further examples and instructions on how to use them, can be found in *my_examples*.

LABVIEW ROBOTIC USER INTERFACE

Manchester Robotics ltd. provide for this a *LabVIEW Robotic User Interface* (LVRUI) that includes the necessary libraries for simulating the Puzzle-Bot in its basic configuration and with the included sensors and actuators such as sonar, reflectance sensors and servo motors. At the same time, it provides the user the ability to connect and seamlessly test their own programs directly with the Puzzle-Bot.

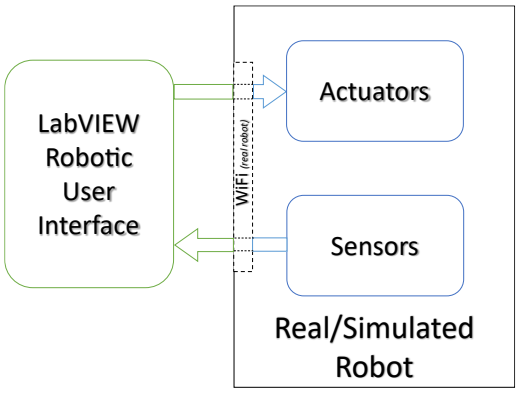


FIGURE 90 MATLAB ROBOTIC USER INTERFACE (MRUI)

LabVIEW Robotic User Interface by *Manchester Robotics ltd.* can be used in a multiple variety of ways depending on the user needs.

WIFI COMMUNICATION

In this section, some basic information and set up when working with the real robot is described.

- The LVRUI communication with the real robot is based on TCP/IP and UDP WiFi protocol.
- The user might experience some communication lag because of WiFi network overload.
- Make sure that no other software interferes with the network (browser, etc.)
- Close the webpage of the robot (don't use them in parallel).
- Try keeping a Line of sight between the robot and the computer (preferable).
- When implementing real time controllers, keep in mind, that the latency between LabVIEW and the microcontroller is around 20 mS depending on the computer performance.

HOW TO USE THE LVRUI

1. **Unzip** the folder named "PuzzlebotLabviewTemplateV1_2.zip" into a directory.
2. Open the LabView Project called "PuzzlebotLabview.lvproj" you should see the following

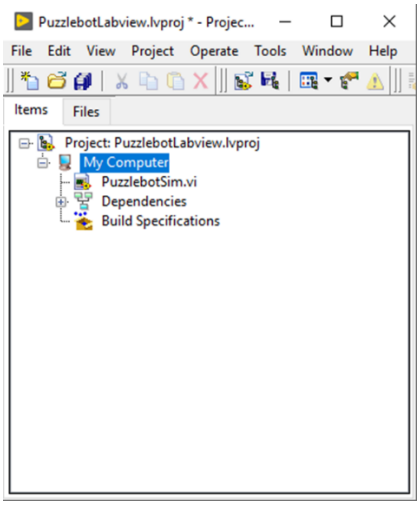


FIGURE 91 LABVIEW PROJECT

3. Open the "PuzzlebotSim.vi", the following screen will be shown

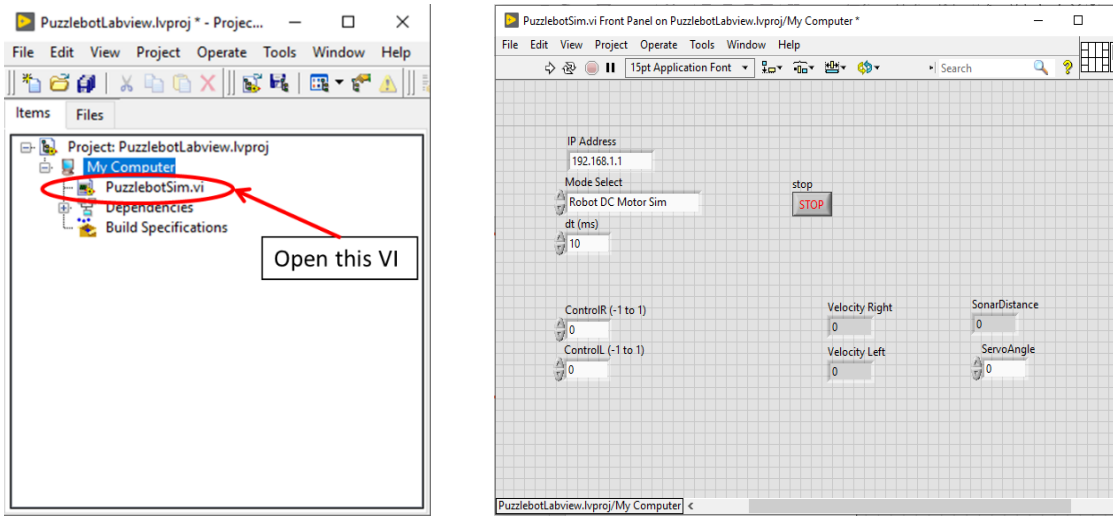


FIGURE 92 PUZZLE-BOT PROJECT

4. Parameters used in Front Panel of the VI

Parameter in LabView	Default Value	Description
IP Address	192.168.1.1	IP Address of the real robot shown on the screen (For this coursework will not be used)
Mode Select	Robot Simulation	Mode Selection Control

		1. Robot Simulation: Simulates the real robot, opens a new window that plots the robot movement and the trajectory that follows. 2. Robot DC Motor Sim: Simulates two DC motors. 3. Real Robot: Used to connect to the real robot. (Not used in this coursework)
dt(ms)	50	Sampling time selection in mS (milliseconds)
ControlR	0	PWM signal percentage applied to the right DC motor. The values can vary in the interval [-1, 1], i.e., [-1,…,-0.5,…,0,…,0.5,…1], where 1 is full power to the motor and -1 is full power in reverse direction.
ControlL	0	PWM signal percentage applied to the left DC motor. The values can vary in the interval [-1, 1], i.e., [-1,…,-0.5,…,0,…,0.5,…1], where 1 is full power to the motor and -1 is full power in reverse direction.
Velocity Right		Indicator showing the right motor velocity
Velocity Left		Indicator showing the left motor velocity
Sonar Distance		Indicator showing the distance measured by the Sonar Sensor (Real Robot only, if included)
Servo Angle		Control the Servo Motor angle on the interval [-90, 90] deg
Stop		Stop button (This button must be used to stop simulation and/or real robot usage)

5. Block Diagram description

   The block diagram can be divided into three sections, Initialisation, Simulation Execution and Close Communication / Simulation End as follows.

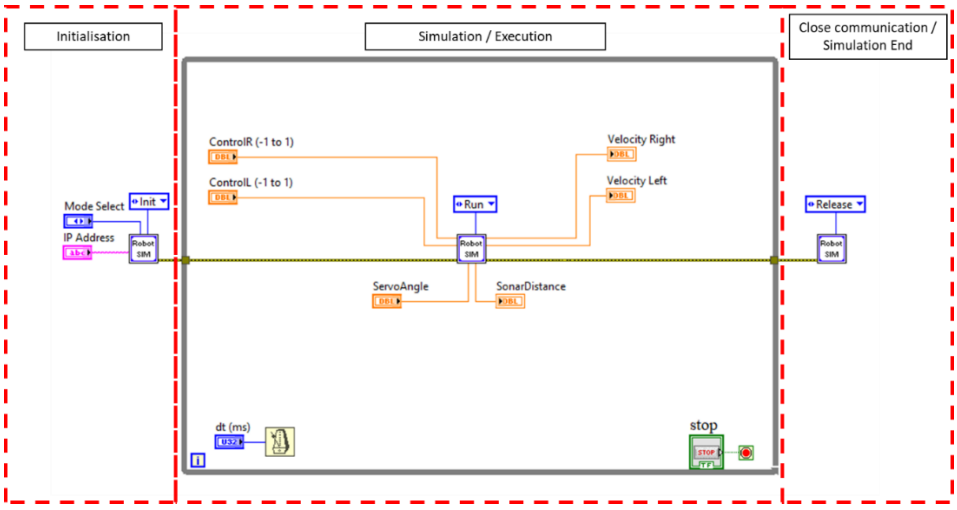


FIGURE 93 BLOCK DIAGRAM

The block diagram uses a SubVi called "Robot SIM" that simulates the dynamics of the robots including the motors.

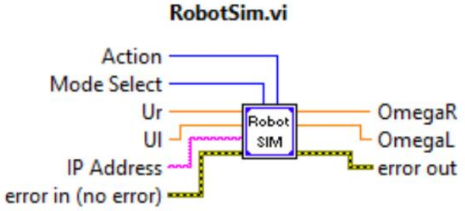


FIGURE 94 SUB-VI

The following table will describe the parameters that will be used in this laboratory.

Parameter	Input / Output	Definition
Mode Select	Input	Mode Selection Control **1.** Robot Simulation: Simulates the real robot, opens a new window that plots the robot movement and the trajectory that follows. **2.** Robot DC Motor Sim: Simulates two DC motors. **3.** Real Robot: Used to connect to the real robot.
Action	Input	Define the action of the SubVi to initialise variables, Run the simulation, or finalise "close" the simulation/ communication.
Ur	Input	PWM signal percentage applied to the right DC motor. The values can vary in the interval [-1, 1], i.e., [-1,…,-0.5,…,0,…,0.5,…1], where 1 is full power to the motor and -1 is full power in reverse direction.
Ul	Input	PWM signal percentage applied to the left DC motor. The values can vary in the interval [-1, 1], i.e., [-1,…,-0.5,…,0,…,0.5,…1], where 1 is full power to the motor and -1 is full power in reverse direction.
IP Address	Input	IP Address used to communicate with the real robot.
error in	Input	Input error function.
OmegaR	Output	Output the right motor velocity as given by the encoders.
OmegaL	Output	Output of the Left motor velocity as given by the encoders.
error out	Output	Error output.

EXAMPLES

LabVIEW Robotics User Interface examples with the simulator and VI templates provided.

Some premade examples are provided to help the user familiarise and understand the capabilities of the LabVIEW PuzzleBot environment.

EXAMPLE 0- Familiarisation with the simulator and VI templates provided.

For this task you can modify the block diagram to

1. Change the "Mode Select" to "Robot DC Motor Sim"
2. Input different values in the interval [-1,1] into "ControlR" and "ControlL".
3. Observe the behaviour of the actuators in "Velocity Right" and "Velocity Left".

Observation 1: The input signal represents the percentage of maximum voltage applied to the DC motor, and the output signal represents the angular velocity of the wheel in radians per second.

Observation 2: Try plotting the outputs and make sure they make sense according to the applied voltage.

Observation 3: USE the STOP button to stop the simulation 

. DO NOT use the "Abort Execution" button  from the task bar (upper bar). This applies for the rest of the tasks.

Example 1- Simple Plotting example for the left and right motor angular velocities.

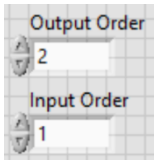
a. Open the project "PuzzlebotLabview.lvproj"
b. Open and run the example "Example_Simple_Plot"
    i. Observe the properties and block diagram to get an initial idea on how to plot data on the LabVIEW environment.
c. Use the PuzzleBot (real robot) with this example. Follow the next steps or the PuzzleBot manual to connect to the robot.
    i. Connect to the robot via WiFi
    ii. Make sure the IP address shown in the robot screen and in the "IP Address" control to match.
    iii. Run the example.

Example 2- In this example the robot drives in a square using an open loop control, based on time.

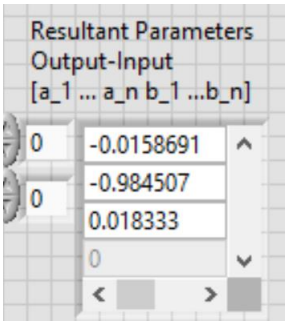
a. Open the project "PuzzlebotLabview.lvproj".
b. Open and run the "Example_Robot_Simulation".
    i. Open the block diagram and observe the control algorithm implemented.
    ii. Can this algorithm be improved? How? What would you do?
c. Use the Puzzle-Bot (real robot) with this example. Follow the next steps or theones in section Connecting to the Puzzle-Bot
    i. Connect to the robot via WiFi
    ii. Make sure the IP address shown in the robot screen and in the "IP Address" control to match.
    iii. Run the example.

Example 3- System Identification example. For this example, the one of the motors (right motor) will be identified using an ARX model offline. For this case, a PRBS (pseudo random binary signal) is used as input for the system.

a. Open the project "PuzzlebotLabview.lvproj".
b. Open and run the "Example_Sys_ID".
c. Change the "output order" and "input order" for the ARX model.



d. Once the simulation finishes, the resultant parameters will be shown as follows



Where the parameter vector $\theta = [a_1 \dots a_n \; b_1 \dots b_n]^T$.

e. Use the PuzzleBot (real robot) with this example. Follow the next steps or the PuzzleBot manual to connect to the robot.
    i. Connect to the robot via WiFi
    ii. Make sure the IP address shown in the robot screen and in the "IP Address" control to match.
    iii. Run the example.

Note The sampling time might need to be changed depending on where the robot is placed, since the friction and slippage due to the surface will alter its dynamical behaviour.

PUZZLE-BOT / ROS

Puzzle-Bot has the capability to be controlled using ROS (Robot Operating System). The current version supported by Puzzle-Bot and its ROS based simulators is *ROS Melodic*.

BASIC CONNECTION DIAGRAMS

Puzzle-Bot connection diagrams using ROS can be depicted in Figure 95 and Figure 97

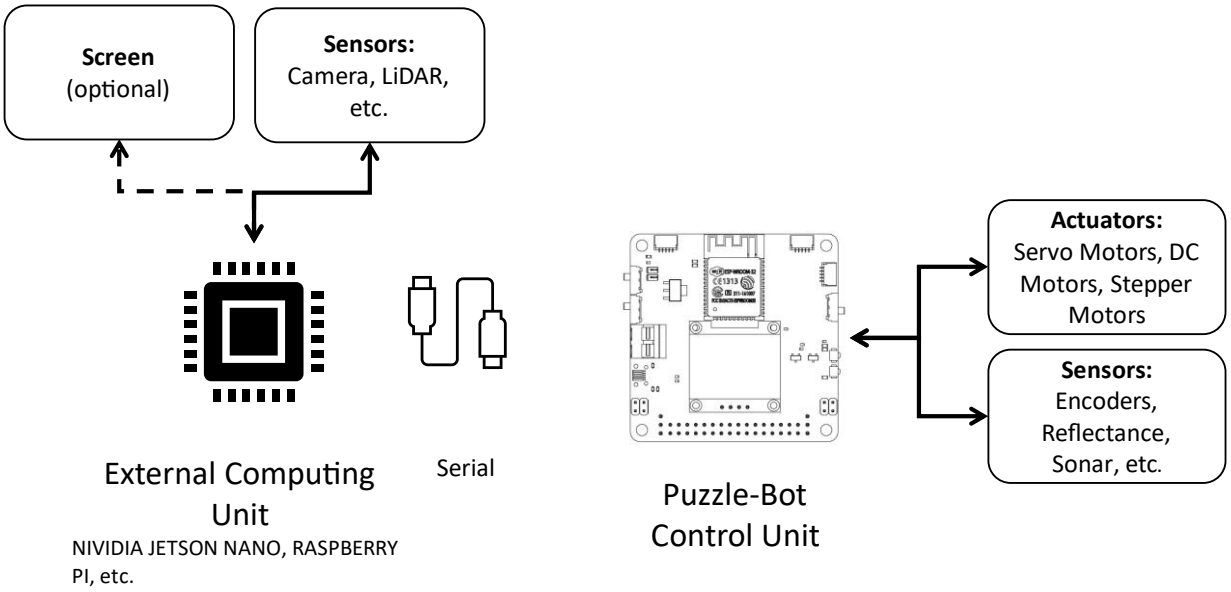


FIGURE 95 PUZZLE-BOT/ROS SERIAL COMMUNICATION CONNECTION DIAGRAM

The configuration shown in Figure 95 consists only of a serial connection between the External Computing Unit and the Puzzle-Bot Control Unit. The communication is done via USB serial communication. For this configuration is advised to use a screen connected to the External Computing Unit or any other communication method such as SSH, to start ROS and the necessary nodes for the robot to work.

The diagram shown in Figure 96 depicts the ROS nodes running inside each of the components shown in this configuration. It can be observed that the External computing unit contains the ros master and other nodes that can be defined by the user, and the Puzzle-Bot Control Unit runs a node that communicates via USB serial to the master and handles the control and communication with the robot sensors and actuators.

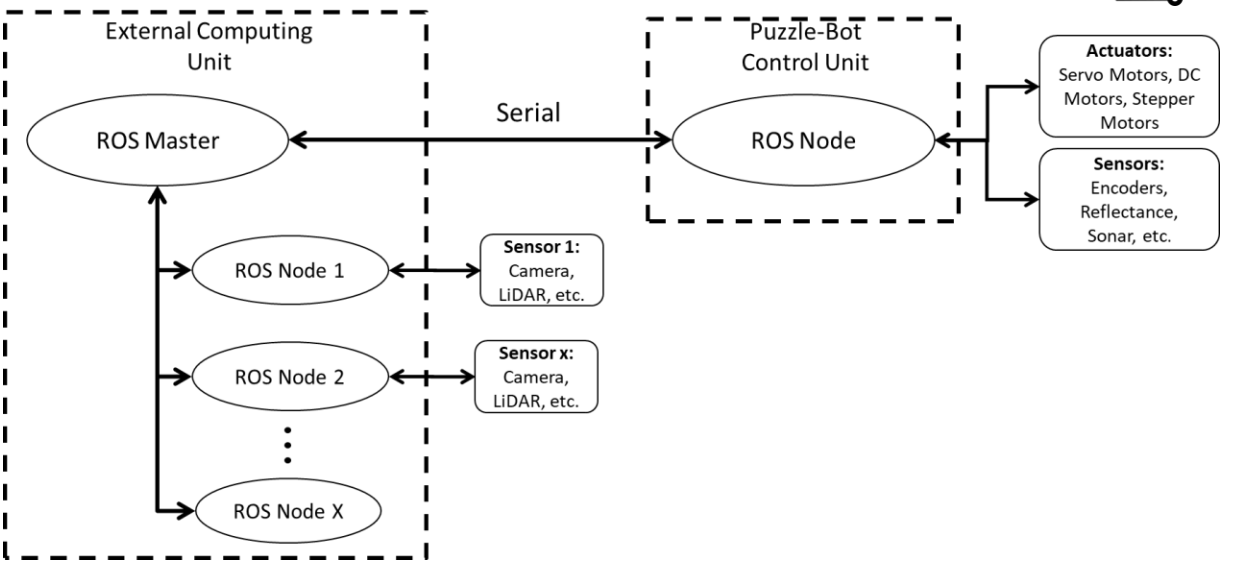


FIGURE 96ROS SERIAL COMMUNICATION

The required configuration and package installation can be found in the section Installing the OS. Furthermore, the steps required for the robot to work in this configuration can be found in section Puzzle-Bot ROS Serial Connection.

The second configuration option (Client ROS Connection Diagram) is depicted in Figure 97. This configuration works as the previous one, with the difference that in this case the user can connect to the External computing unit (ROS Master) via Wi-Fi.

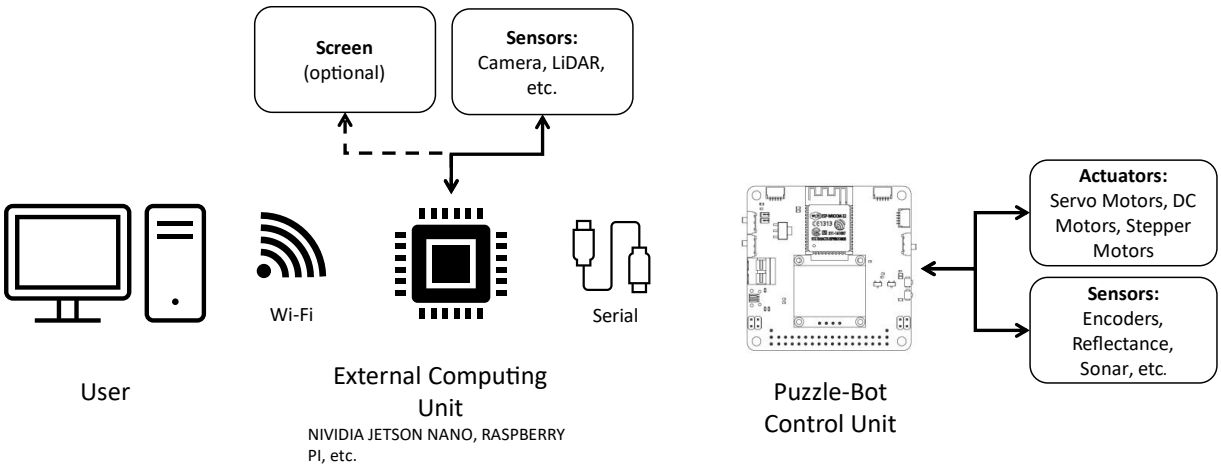


FIGURE 97 CLIENT ROS CONNECTION DIAGRAM

From a ROS perspective this configuration can be seen in Figure 98

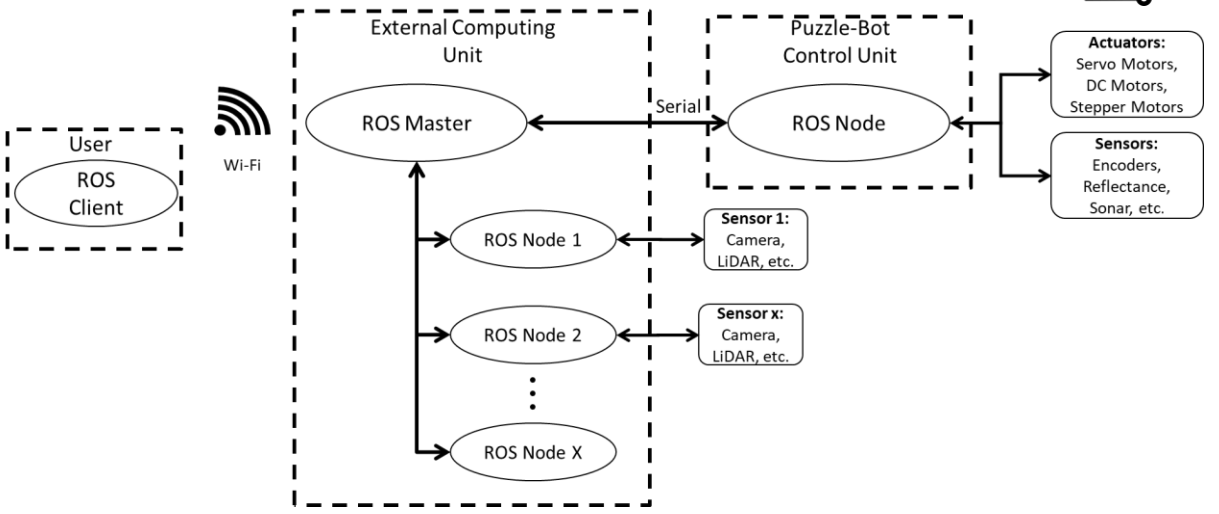


FIGURE 98 CLIENT ROS CONNECTION DIAGRAM

This configuration requires the user to set up a wireless local area network (WLAN) between the User and the External computing unit, a hotspot, or an ad-hoc network. Please read the official ROS documentation on how to connect to a remote master.

INSTALLING THE OS

- For desktops or laptops, install Ubuntu 18.04 as shown in the following link.

  https://releases.ubuntu.com/18.04/

- For Nvidia Jetson Nano ™ 2Gb Developer Kit follow the instructions in the following link

  https://developer.nvidia.com/embedded/learn/get-started-jetson-nano-2gb-devkit

- For Raspberry Pi follow the steps in the following link

  https://projects.raspberrypi.org/en/projects/raspberry-pi-getting-started

  ***Please note that for other computing units, the installation of the OS may vary.*

INSTALLING ROS MELODIC AND NECESSARY PACKAGES

- Install *ROS Melodic form the official website* as shown in the following link

  http://wiki.ros.org/ROS/Installation

- Install the *ROS Serial Package* on the ROS Workstation as shown in the following link

  http://wiki.ros.org/rosserial_arduino/Tutorials/Arduino%20IDE%20Setup

```
sudo apt-get install ros-melodic-rosserial-arduino
sudo apt-get install ros-melodic-rosserial
```

- Install the *ROS Control Package (ros_control)*

  http://wiki.ros.org/ros_control

```
sudo apt-get install ros-melodic-ros-control ros-melodic-ros-controllers
```

- Install the *ROS Teleop Package (teleop_twist_keyboard)*

  http://wiki.ros.org/teleop_twist_keyboard

```
sudo apt-get install ros-melodic-teleop-twist-keyboard
```

SETTING UP THE CATKIN WORKSPACE

- Unzip the folder *ROS_ Simulator_V1*
- Copy all the folders inside the folder *ROS_ Simulator_V1to* the folder *catkin_ws/src* as shown in the following figure

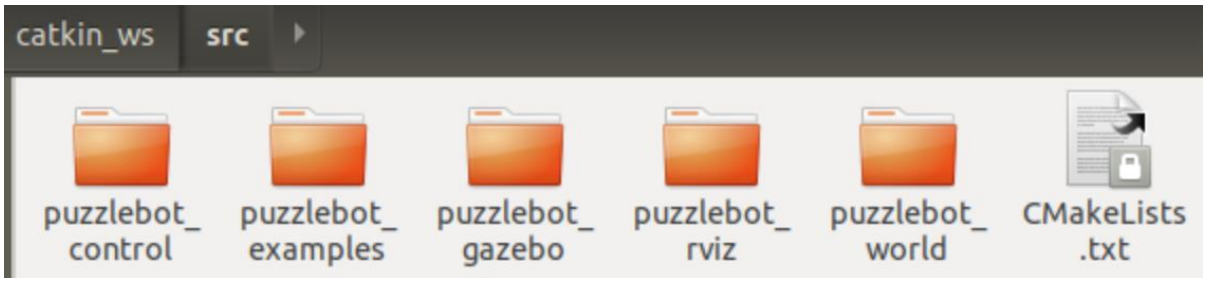


FIGURE 99 CATKIN WORKSPACE

- Compile the *catkin workspace*, by typing into the terminal

```
cd catkin_ws

catkin_make
```

- Source the setup file in the devel directory by typing the following command in each new terminal

```
source devel/setup.bash

**To avoid this step follow the steps in the following website:
https://answers.ros.org/question/117801/how-to-get-the-line-source-
develsetupbash-to-run-after-every-time-you-catkin_make/
```

PUZZLE-BOT – GAZEBO SIMULATION

Manchester Robotics ltd. Has developed a Gazebo simulation for the Puzzle-Bot. The purpose of this simulator is to provide the user the ability to test their code using a Puzzle-Bot dynamical model with perturbations and noise resembling the real robot.

BASIC PUZZLE-BOT SIMULATION IN THE GAZEBO ENVIRONMENT

The Puzzle-Bot gazebo simulator allows the user to test their code before using it with the real robot. To run the gazebo simulator, follow the next steps

- Follow the steps described in sections: *Installing ROS Melodic and necessary Packages* and *Setting Up the Catkin Workspace*.
- Type the following command on a terminal

```
cd catkin_ws

source devel/setup.bash

roslaunch puzzlebot_gazebo puzzlebot_gazebo.launch
```

- The following figure should automatically appear

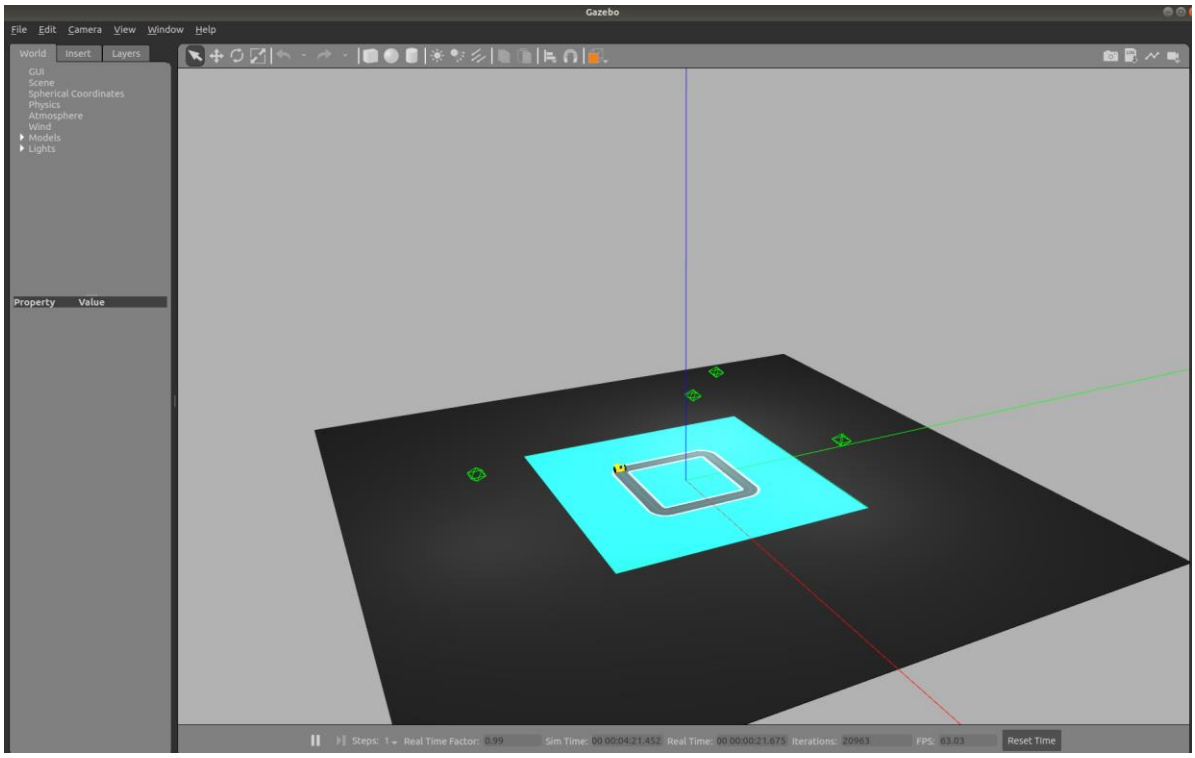


FIGURE 100 PUZZLE-BOT GAZEBO SIMULATION

SENDING COMMANDS TO THE ROBOT

The topics to subscribe or publish to the robot can be seen by following the next steps

- Open a new terminal and type

```
rostopic list
```

- The topic used to send command to the robot is `/cmd_vel` and the type of message is `geometry/msgs/Twist`. Where the linear velocity v and angular velocity ω are set as follows (refer to Figure 101).

```
rostopic pub /cmd_vel geometry_msgs/Twist "linear:
x: v
y: 0.0
z: 0.0
angular:
x: 0.0
y: 0.0
z: ω"
```

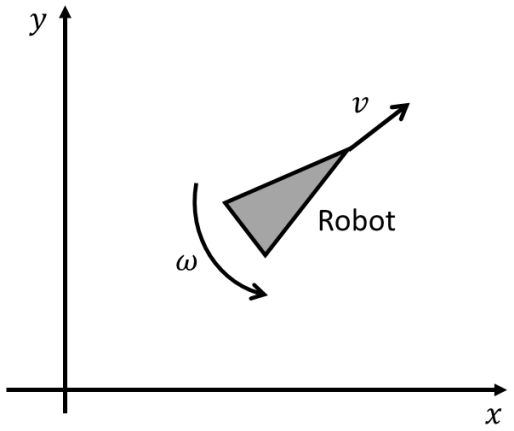


FIGURE 101ROBOT LINEAR AND ANGULAR VELOCITIES

- An example command to the robot would be

```
rostopic pub /cmd_vel geometry_msgs/Twist "linear: x: 0.3 y: 0.0 z: 0.0
                angular: x: 0.0 y: 0.0 z: 0.5"
```

- The result would be a robot moving in a circle.

ROBOT SIMULATED WHEEL VELOCITIES

- The user can subscribe or echo the topics `/wr` and `/wl` to get the angular velocities of each of the wheels, by opening a new terminal and typing

```
rostopic echo /wr

rostopic echo /wl
```

- The user can observe these velocities using the `rqt_plot` as follows
    - Run in a new terminal

```
rosrun rqt_plot rqt_plot
```

    - Type in the Topic `/wr` or `/wl` and add it to the plot using the + sign

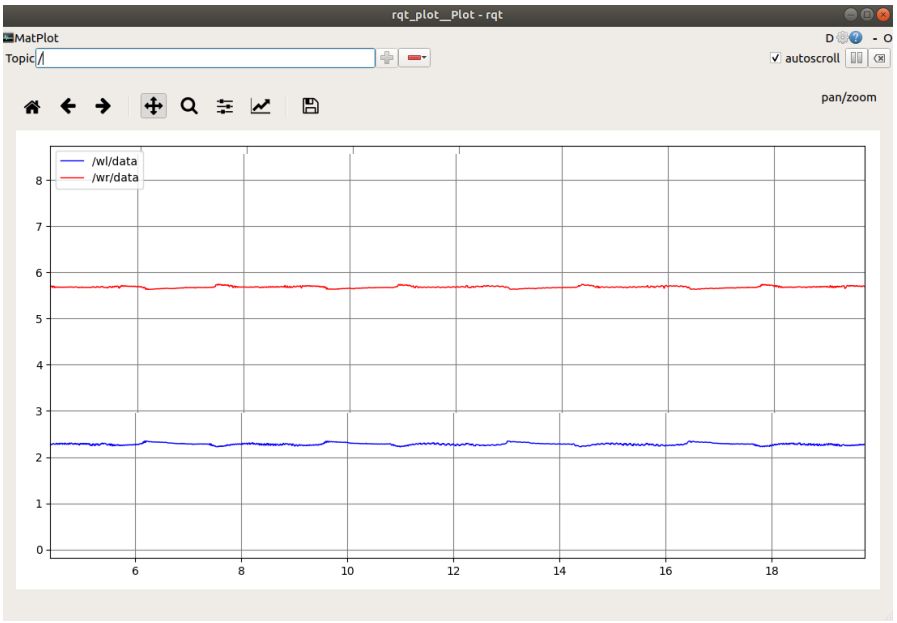


FIGURE 102WHEEL VELOCITIES AND RQT PLOT

GAZEBO-CAMERA SIMULATION

- The user can activate the camera by going into the Gazebo environment `Window>>Topic Visualization (Ctrl+T)`
    - A pop-up window will appear

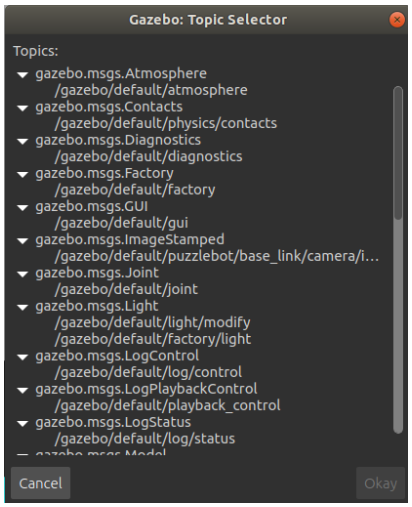


FIGURE 103GAZEBO TOPIC SELECTOR

- Select the topic

  `/gazebo/default/puzzlebot/base_link/camera/image_raw`

- The topic is located under `gazebo.msgs.ImageStamped`
- The following pop-up window should appear

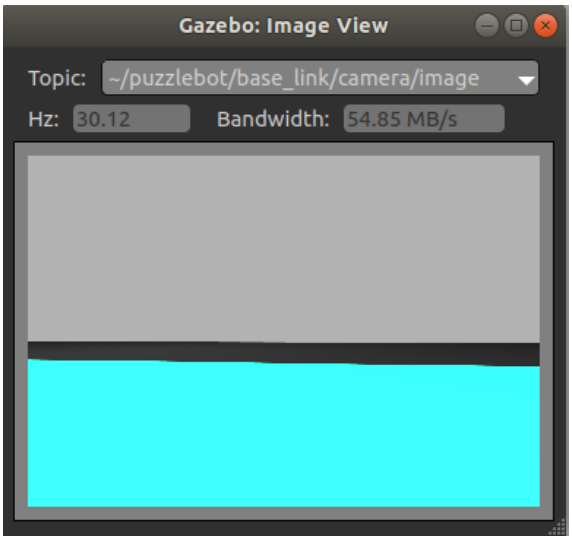


FIGURE 104GAZEBO CAMERA

- The user can subscribe to the camera via one of the following topics

  `/camera/image_raw`

  `/camera/image_raw/compressed`

GAZEBO EXAMPLES

Manchester Robotics ltd, has compiled some basic examples using the gazebo simulator. These Examples include a teleoperated Puzzle-Bot simulation and a basic position control algorithm.

PUZZLE-BOT TELEOPERATED

This example shows how the Puzzle-Bot Gazebo simulation can be teleoperated

- Open a new terminal
- Run the following command

```
roslaunch puzzlebot_examples teleop_gazebo.launch
```

**Make sure the steps in section "Installing ROS Melodic and necessary Packages" are done*

- The following screen should appear

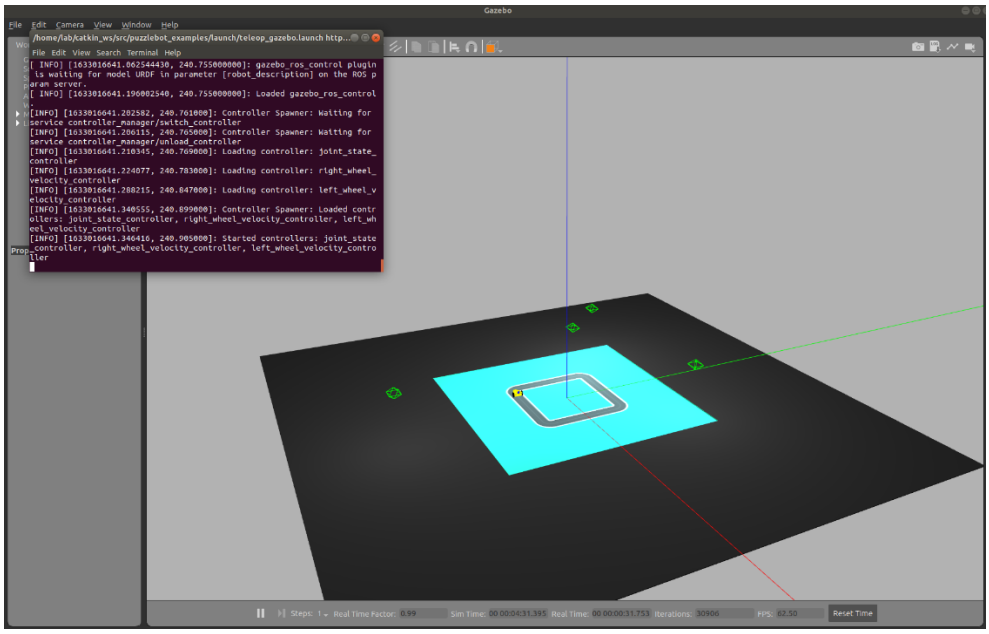


FIGURE 105PUZZLE-BOT GAZEBO TELEOPERATION EXAMPLE

- Use the following keys to move the robot in different directions

```
Reading from the keyboard  and Publishing to Twist!
Moving around:
   u    i    o
   j    k    l
   m    ,    .
q/z: increase/decrease max speeds by 10%
w/x: increase/decrease only linear speed by 10%
e/c: increase/decrease only angular speed by 10%
anything else: stop
CTRL-C to quit
```

- More information about the teleoperation package can be found in the following link http://wiki.ros.org/teleop_twist_keyboard

PUZZLE-BOT BASIC CONTROL ALGORITHM

This example comprises of a basic position controller. For this example, the robot moves in a square around the centre of map.

To run the example, follow the next steps

- Open a new terminal
- Run the following command

```
roslaunch puzzlebot_examples square_gazebo.launch
```

**Make sure the steps in section "Installing ROS Melodic and necessary Packages" are done*

- The following screen should appear

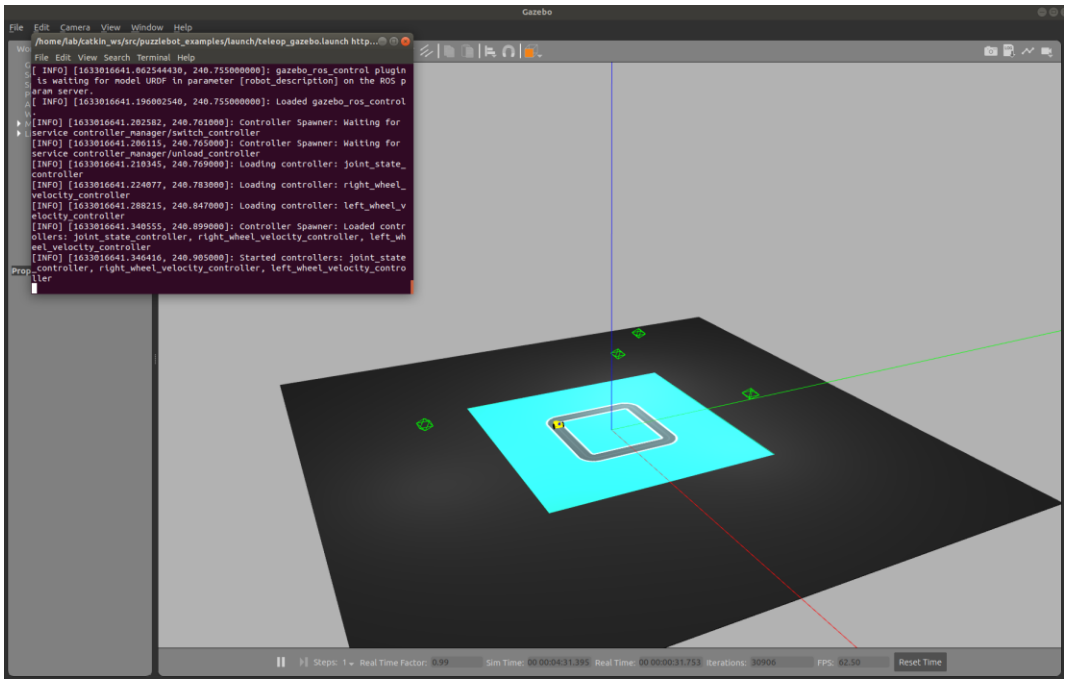


FIGURE 106PUZZLE-BOT POINT STABILISATION

- The robot will start moving automatically in a square. The user can activate the camera by following the steps in section Gazebo-Camera Simulation.

The controller is based on a basic point stabilisation algorithm [1], as shown inFigure 107. For the case of a nonholonomic robot like the Puzzle-Bot, let the robot pose to be represented by the vector $\boldsymbol{\rho}_r = \begin{pmatrix} x_r \\ y_r \\ \theta_r \end{pmatrix}$.

The inputs are the linear and angular velocity of the robot $\boldsymbol{V}_r = \begin{pmatrix} v_r \\ \omega_r \end{pmatrix}$. The linear velocity of the robot, v_r, is always oriented in the direction of X_r of the robot reference frame because of the non-holonomic constraint.

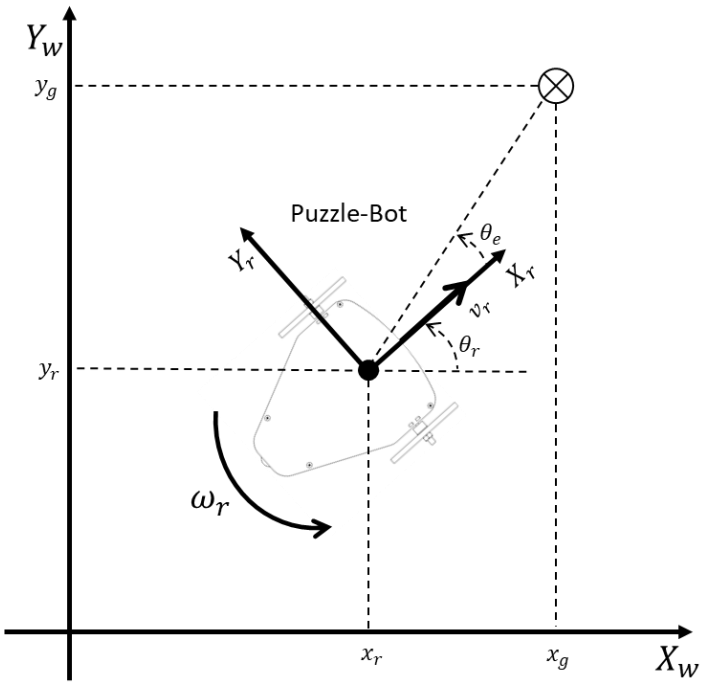


FIGURE 107ROBOT REFERENCE FRAME

The error between the goal and the robot pose (odometry) is computed as in the following:

$$e_x = x_g - x_r$$

$$e_y = y_g - y_r$$

$$e_\theta = atan2(e_y, e_x) - \theta_r$$

The robot pose for this example is obtained using dead reckoning localisation. The controller can then be defined as follows

$$v_r = K_d e_d$$

$$\omega_r = K_\theta e_\theta$$

Where $e_d = \sqrt{e_x^2 + e_y^2}$ and K_d and K_θ are positive constant parameters to be chosen by the user.

PUZZLE-BOT DYNAMICAL SIMULATION USING RVIZ

A dynamical simulator for the Puzzle-Bot, based on [2], has been developed by Manchester Robotics. This simulation does not require the usage of Gazebo and can be used to simulate and implement different types of controllers or to study and evaluate the dynamical behaviour of the robot.

PUZZLE-BOT DYNAMICAL SIMULATION

The simulated robot can be launched using the following steps

- Open a terminal
- (Optional) The mode of operation of this simulation can be configured using the YAML file located in *puzzlebot_rviz/configpuzzlebot_sim.yaml.* Inside this file the user can configure the following:
    - Different modes of operation for the inputs to the robot as shown in the following table

Mode	Description	Topic Name and Type	Topic Information
1	Linear and Angular Velocities (PID control)	/cmd_vel geometry_msgs/Twist	linear x: Linear speed of the robot y: Not used z: Not Used angular: x: Not Used y: Not Used z: Angular Speed of the robot
2	Wheel angular velocities setpoint (PID control)	/cmd_wL, /cmd_wR std_msgs/Float32	data: Control Set Point for wheel velocities
3	Wheel PWM voltage signal	/cmd_pwmL, /cmd_pwmR std_msgs/Float32	data: PWM duty cycle for each motor [-1,1]

    - The topic names
    - The PID controller parameter for the wheels in case the mode of operation is properly configured.
- Run the following command

```
roslaunch puzzlebot_rviz puzzlebot_sim.launch
```

**Make sure the steps in section "Installing ROS Melodic and necessary Packages" are done*

- The topics used to send commands to the robot are dependant on the YAML file configuration located in *puzzlebot_rviz/configpuzzlebot_sim.yaml. The default command is set to /puzzlebot_sim1/cmd_vel.*
- The topics used to subscribe and receive the information from the simulation are the following
    - */puzzlebot_sim1/odom :* Displays the odometry information of the robot.
    - */puzzlebot_sim1/puzzlebot/motorX/eta :* Displays the velocity of the motors.
    - */v and /w :* Display the information about the robot linear and angular velocities.

PUZZLE-BOT SIMULATOR USING RVIZ

Manchester Robotics has configured an RVIZ viewer for the user to be able to see the Puzzle-Bot while using the simulator described in the previous section. This visualisation tool can be used by following the next steps

- Open a terminal
- Run the following command

```
roslaunch puzzlebot_rviz puzzlebot_rviz.launch
```

- The following screen should appear

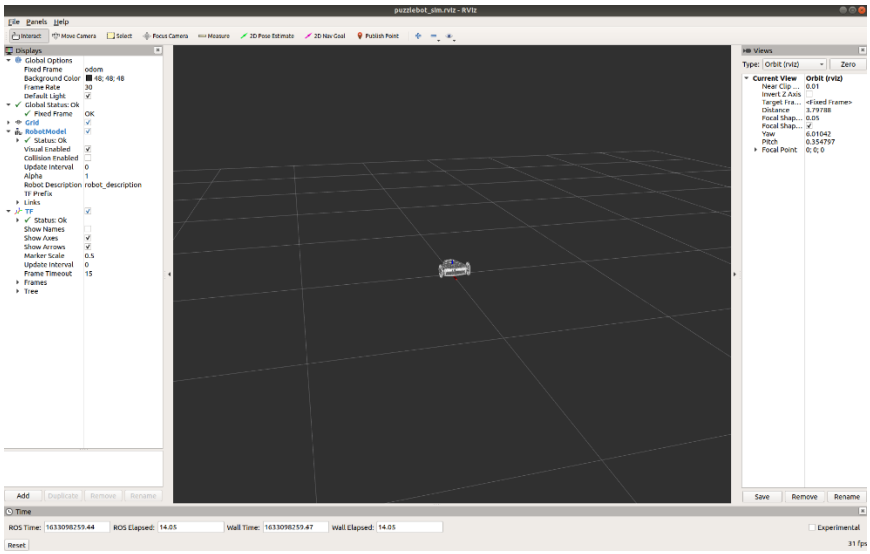


FIGURE 108PUZZLE-BOT RVIZ SIMULATION

- To control the robot, please refer to the previous section Puzzle-Bot Dynamical Simulation.

PUZZLE-BOT SIMULATION EXAMPLES

Manchester Robotics ltd, has compiled some basic examples using the dynamical simulator. These Examples include a teleoperated Puzzle-Bot simulation and a basic position control algorithm.

TELEOPERATED PUZZLE-BOT EXAMPLE

This example shows how the Puzzle-Bot Gazebo simulation can be teleoperated

- Open a new terminal
- Run the following command

```
roslaunch puzzlebot_examples teleop_gazebo.launch
```

**Make sure the steps in section "Installing ROS Melodic and necessary Packages" are done*

- The screen in figure Figure 108 should appear
- Use the following keys to move the robot in different directions

```
Reading from the keyboard  and Publishing to Twist!
Moving around:
   u    i    o
   j    k    l
   m    ,    .
q/z: increase/decrease max speeds by 10%
w/x: increase/decrease only linear speed by 10%
e/c: increase/decrease only angular speed by 10%
anything else: stop
CTRL-C to quit
```

- More information about the teleoperation package can be found in the following link http://wiki.ros.org/teleop_twist_keyboard

PUZZLE-BOT BASIC CONTROL ALGORITHM

This example comprises of a basic position controller. For this example, the robot moves in a square around the centre of map.

To run the example, follow the next steps

- Open a new terminal
- Run the following command

```
roslaunch puzzlebot_examples square_rviz.launch
```

**Make sure the steps in section "Installing ROS Melodic and necessary Packages" are done*

- The screen in figure Figure 108 should appear.
- The robot should start moving automatically in a square.
- More information about the control algorithm can be found in section Puzzle-Bot Basic Control Algorithm.

PUZZLE-BOT ROS SERIAL CONNECTION

Puzzle-Bot has the capability of acting as ROS Node, as of Manchester Robotics ltd. Binaries V1.7.

To use the Puzzle-Bot with ROS, follow the following steps.

1. Log into the Puzzle-Bot web interface as described in section Basic Web Interface
2. Go to the configuration file and make sure the ROS parameter is enabled in the configuration file as stated in section Configuration File (YAML file)

```
# Active modules settings
ActiveModules:
  Servo:   1                    # Servo motor on/off (1/0)
  Sonar:   0                    # Sonar on/off (1/0)
  Reflect: 0                    # Reflectance sensor on/off (1/0)
  Lidar:   0                    # Lidar on/off (1/0)
  Screen:  1                    # Screen on/off (1/0)
  Ros:     1                    # Ros communication on/off (1/0)
```

FIGURE 109: PUZZLE-BOT FILE CONFIGURATION

3. Restart the PuzzleBot.
4. Close PuzzleBot Web Interface
5. Connect the USB cable between the Puzzle-Bot and the external computing unit running ROS to the PuzzleBot (NVIDIA Jetson NANO, RPi, etc.)

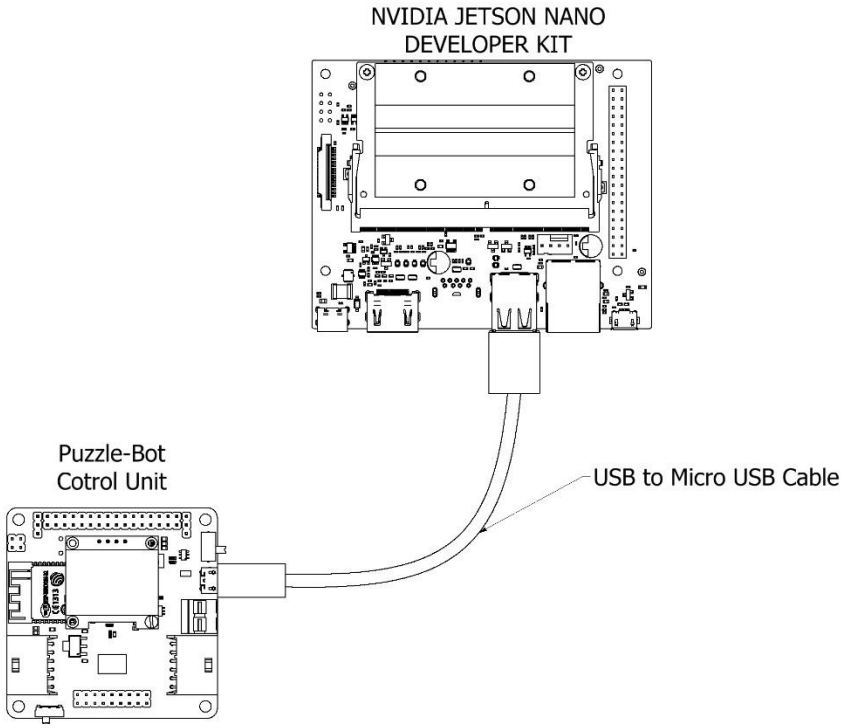


FIGURE 110 CONNECTION EXAMPLE BETWEEN NIVIDA JETSON AND PUZZLE-BOT CONTROL UNIT

6. Verify which port is the Puzzle-Bot Control Unit Connected to, running the following commands

```
cd /dev
  ls
```

7. Verify the port where the Puzzle-Bot Control Unit is connected, should appear as

```
/dev/ttyUSBx
```

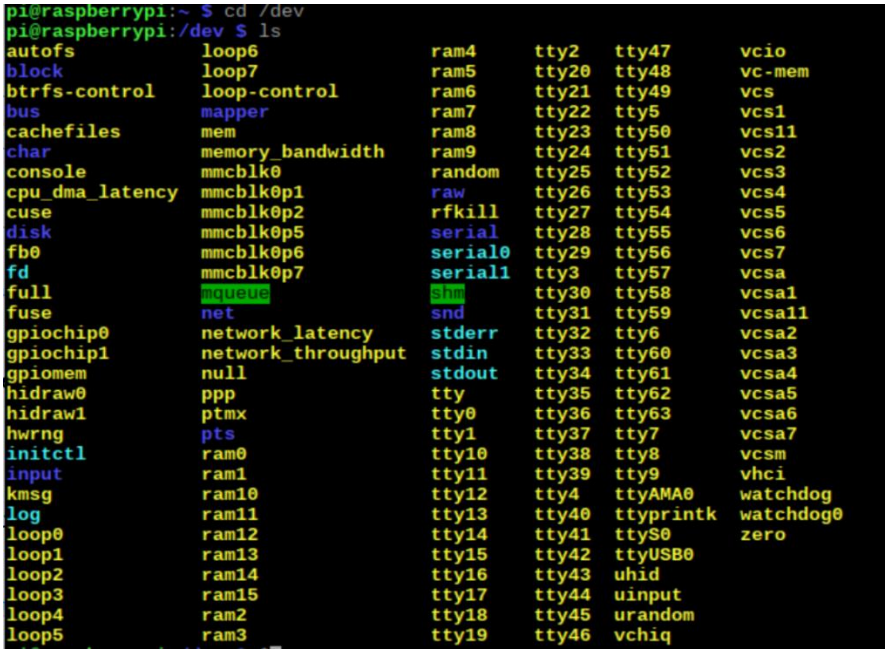


FIGURE 111 USB PORTS

8. Give permission to the USB ports in ubuntu with the following command (where x represents the USB port connected)

```
sudo chmod 666 /dev/ttyUSBx
```

*This can be done permanently by adding the user to the DIALOUT https://askubuntu.com/questions/58119/changing-permissions-on-serial-port

9. Launch the roscore in a new terminal window.

```
roscore
```

```
mcr2@mcr2-desktop:~$ sudo chmod 666 /dev/ttyUSB0
[sudo] password for mcr2:
mcr2@mcr2-desktop:~$ roscore
... logging to /home/mcr2/.ros/log/842660a0-000c-11ec-af64-02425500e0f6/roslaunch-mcr2-desktop-6860.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://mcr2-desktop:45115/
ros_comm version 1.14.11


SUMMARY
========

PARAMETERS
 * /rosdistro: melodic
 * /rosversion: 1.14.11

NODES

auto-starting new master
process[master]: started with pid [6931]
ROS_MASTER_URI=http://mcr2-desktop:11311/

setting /run_id to 842660a0-000c-11ec-af64-02425500e0f6
process[rosout-1]: started with pid [6945]
started core service [/rosout]
```

FIGURE 112 ROSCORE LAUNCH

10. Next, run the *rosserial* client application in a new terminal window that forwards your Arduino messages to the rest of ROS. Make sure to use the correct serial port.

```
rosrun rosserial_python serial_node.py /dev/ttyUSBx
```

```
mcr2@mcr2-desktop:~$ rosrun rosserial_python serial_node.py
[INFO] [1629281453.131909]: ROS Serial Python Node
[INFO] [1629281453.149065]: Connecting to /dev/ttyUSB0 at 57600 baud
[INFO] [1629281455.261840]: Requesting topics...
[INFO] [1629281455.397646]: Note: publish buffer size is 512 bytes
[INFO] [1629281455.405852]: Setup publisher on wr [std_msgs/Float32]
[INFO] [1629281455.422738]: Setup publisher on wl [std_msgs/Float32]
[INFO] [1629281455.439056]: Note: subscribe buffer size is 512 bytes
[INFO] [1629281455.443982]: Setup subscriber on cmd_wR [std_msgs/Float32]
[INFO] [1629281455.460402]: Setup subscriber on cmd_wL [std_msgs/Float32]
[INFO] [1629281455.473586]: Setup subscriber on servo [std_msgs/Float32]
```

FIGURE 113ROS SERIAL CLIENT

More information about rosserial and rosserial-arduino can be found in

http://wiki.ros.org/rosserial_arduino

11. In a new terminal verify the topics are displayed using the following command

```
rostopic list
```

12. The user can publish or echo from the topics

```
mcr2@mcr2-desktop:~$ rostopic list
/cmd_wL
/cmd_wR
/diagnostics
/rosout
/rosout_agg
/servo
/wl
/wr
mcr2@mcr2-desktop:~$ rostopic pub /cmd_w
/cmd_wL  /cmd_wR
mcr2@mcr2-desktop:~$ rostopic pub /cmd_wL std_msgs/Float32 "data: 0.0"
```

FIGURE 114 PUBLISHING TO A TOPIC

The Topics used for the Puzzle-Bot, will vary depending on the control input selected by the user in the configuration file (more information on the section Parameter Configuration File description).

```
Robot:
  Type: 1                       # 1-differential drive robot; 2-holonomic robot
  ControlInput: 1               # 1-robot linear and angular velocities; 2-wheel angular velocities setpoints; 3-wheel
  WheelBase: 0.083              # Half of the robot width
  WheelRadius: 0.0505           # Wheel radius
  TopicVx: VelocityLinearX      # Topic for receiving linear velocity of the robot
  TopicVy: VelocityLinearY      # Topic for receiving linear velocity of the robot on Y(for holonomic robot)
  TopicW: VelocityAngular       # Topic for receiving angular velocity of the robot
```

FIGURE 115 CONFIGURATION FILE CONTROL INPUT

TABLE 11 ROS TOPIC DESCRIPTION

Selection	Description	Topic Name and Type	Topic Information
1	Linear and Angular Velocities	/cmd_vel geometry_msgs/Twist	linear x: Linear speed of the robot y: Not used z: Not Used angular: x: Not Used y: Not Used z: Angular Speed of the robot
2	Wheel angular velocities setpoint (PID control)	/cmd_wL, /cmd_wR std_msgs/Float32	data: Control Set Point for wheel velocities
3	Wheel PWM voltage signal	/cmd_pwmL, /cmd_pwmR std_msgs/Float32	data: PWM duty cycle for each motor [-1,1]
		/servo	data: Servo motor angle

More information about the topics, can be obtained using the commands found in

http://wiki.ros.org/ROS/Tutorials/UnderstandingTopics

```
rostopic bw     display bandwidth used by topic
rostopic echo   print messages to screen
rostopic hz     display publishing rate of topic
rostopic list   print information about active topics
rostopic pub    publish data to topic
rostopic type   print topic type
```

13. The user has access to the velocities given by the left and right encoders using the following ROS command respectively

```
rostopic echo /wl
rostopic echo /wr
```

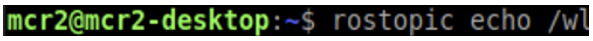


FIGURE 116 ROSTOPIC EXAMPLE FOR GETTING WHEEL SPEED

14. The user can now use these topics and have fun creating their own ROS programs.

CLIENT ROS CONNECTION (NVIDIA JETSON NANO)

In this configuration, the user can connect remotely the PC as a client of the ROS master running in the external computing unit as shown in Figure 97. For this configuration, the user must set up a hotspot to run inside the external computing unit, where the ROS Master is running, and connect the clients to the different Puzzle-Bot topics.

SETTING UP A HOTSPOT FOR ROS MASTER

In this section a basic hotspot configuration using the NVIDIA Jetson Nano will be described.

Firstly, it is important to remember how ROS networking works. As pictured in Figure 98, all wireless devices communicate connecting to a wireless network. So, it is crucial to set the parameters of the network correctly in all devices. Not doing this will create communication problems, such as being able to visualise topics but not being able to send information between them.

There are three points to following

SET UP A HOTSPOT

1. Log in to you into the NVIDIA Jetson Nano. You would need a screen, keyboard, and mouse.

This will look very similar to ubuntu.

2. On the right lower corner next to the clock, click on Network Connections. A window will pop up and click on the + symbol to create a new network

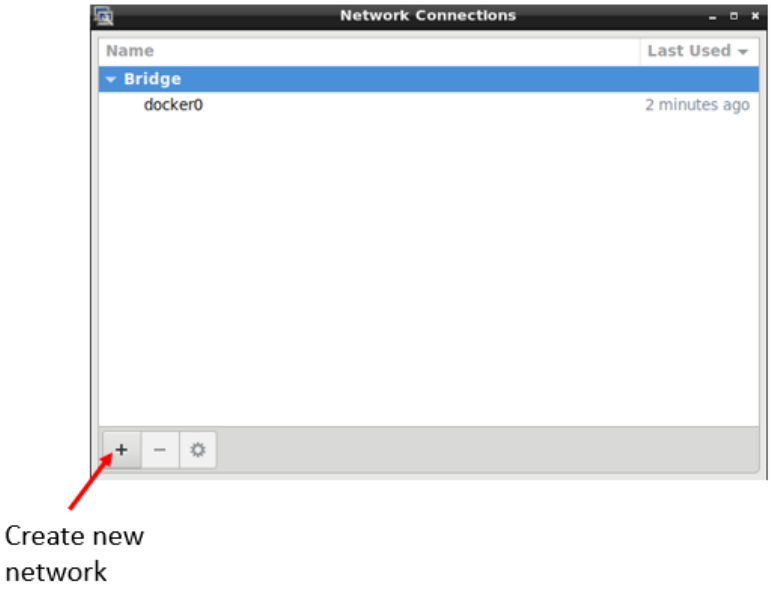


FIGURE 117 CREATING A NEW NETWORK

3. Personalise the Network connection name and SSID and select Hotspot mode.

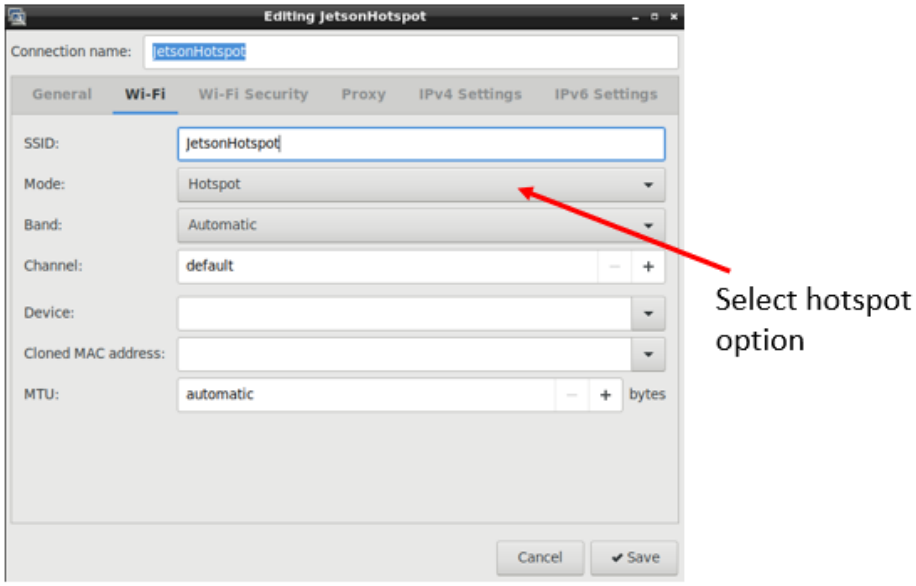


FIGURE 118 PERSONALISE HOTSPOT PARAMETERS

4. On the Wi-Fi Security tab, select the security of your choice and set a password.

5. Additionally, you can set a fixed IP, and the IP segment of the network. This is done on the tap IPv4 Settings.

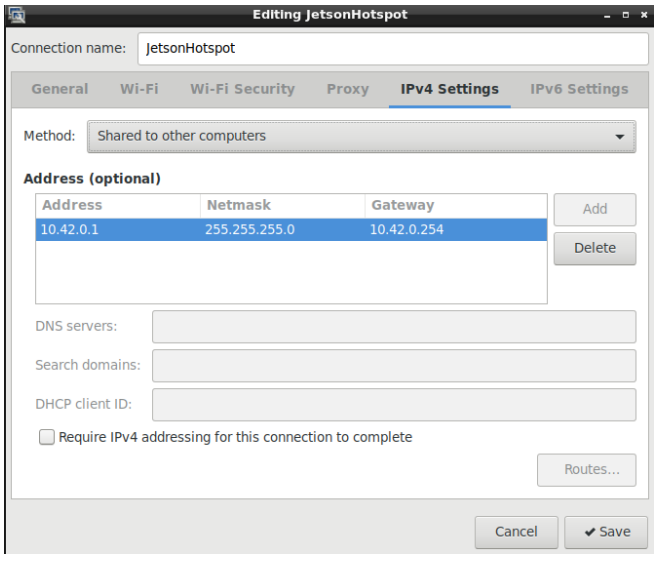


FIGURE 119 SET FIXED IP FOR THE HOTSPOT

6. Then, click save on the lower right button. Once this is done the hotspot would be active every time the robot starts.

CONNECT YOUR COMPUTER TO THE HOTSPOT.

1. Connect your computer to the hotspot, as you would do with any other Wi-Fi network.

It is recommended that every device has its own fixed IP to avoid IP clashes. For that reason,

2. Edit the IP of the device in the hotspot.

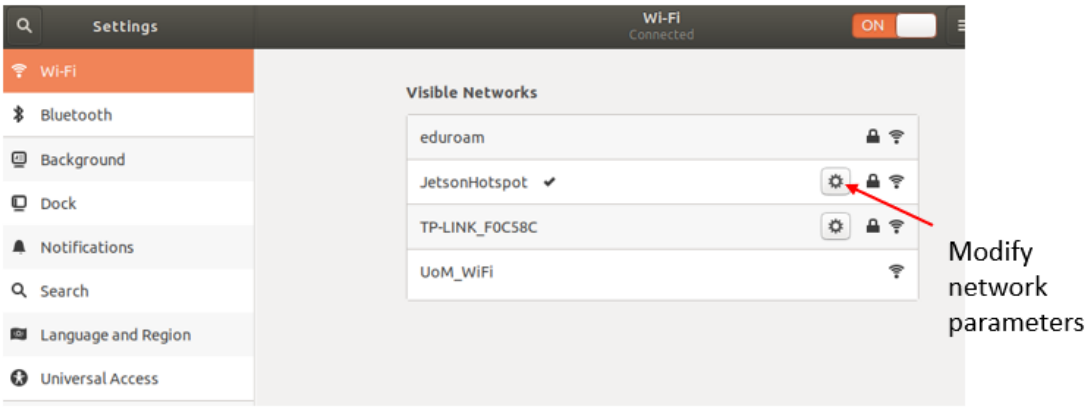


FIGURE 120 EDIT NETWORK SETTING OF THE COMPUTER

3. Set an IP that is not used in the network. In this example the 10.42.0.2 will be selected for this device.

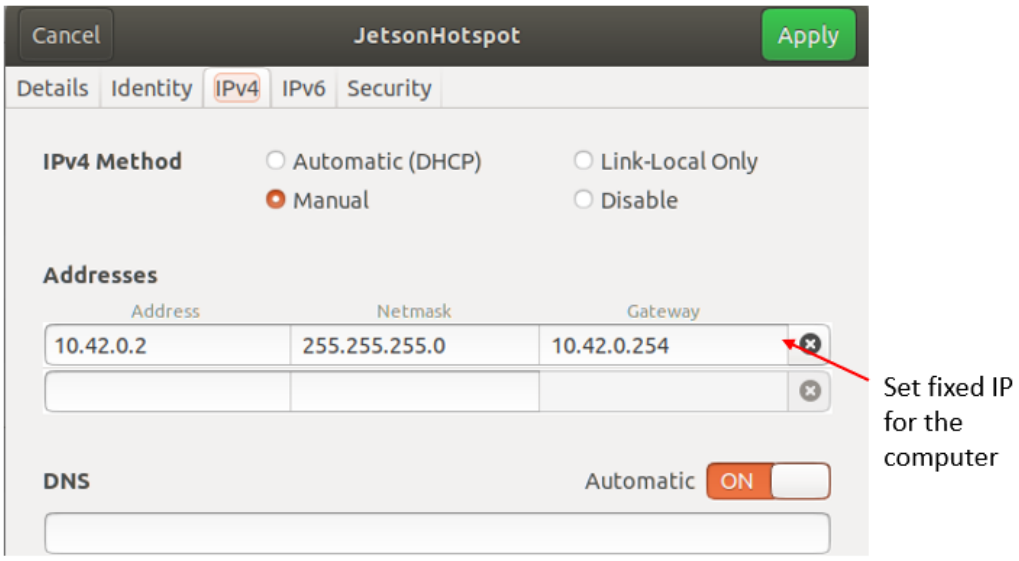


FIGURE 121 SET THE PARAMETERS OF THE IP BASED ON IP OF MASTER

4. Test that both devices communicate correctly, by pining from each computer to the other.

NOTE: If you have Ubuntu 18.04, the firewall is usually enabled by default, which is going to stop the communicating with ROS from working. Please be sure with the following commands that your firewall is disable.

a. You check the status of the firewall using:

```
sudo ufw status
```

b. If the status is active. You can disable it using

```
sudo ufw disable
```

5. Set up the parameters for the ROS Networking.

The parameters to be set in the NVIDIA Jetson are:

```
export ROS_ ROS_MASTER_URI=10.42.0.1

export ROS_IP=10.42.0.1
```

As the hotspot IP will remain fixed, the configuration can be saved in the bashrc file. You can access to the bashrc file (as shown in Figure 122) with the following command:

```
sudo nano ~/.bashrc
```

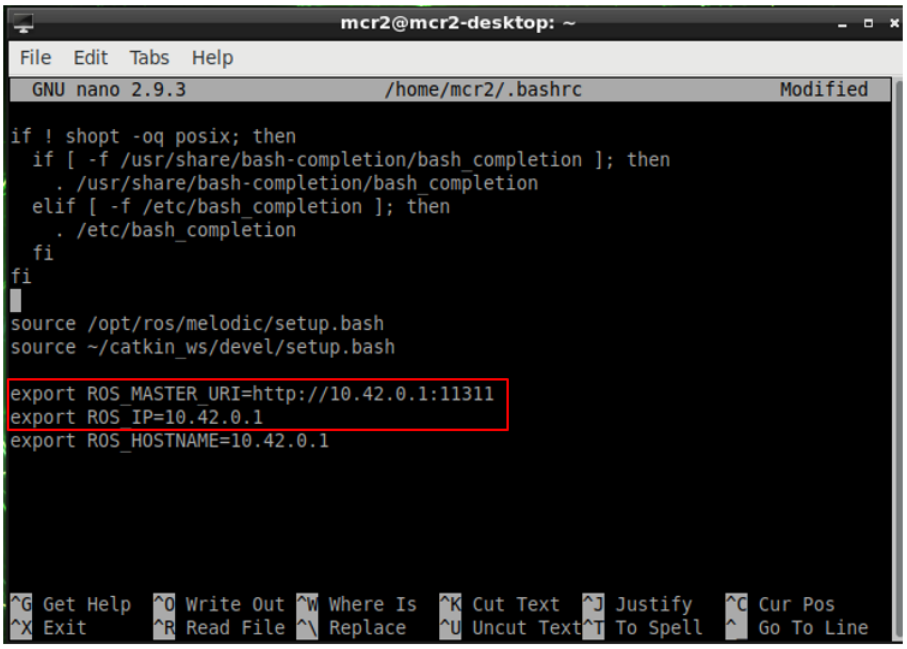


FIGURE 122 PARAMETERS FOR ROS NETWORK IN NVIDIA JETSON FOR

To save the changes, follow the instruction on the bottom of the window.

NOTE: "**^**" means "**control +**". For example, "**^X Exit**" means pressing Control and x at the same time to exit.

NOTE 2: The parameter ROS_HOSTNAME shown in Figure 122 requires an extra configuration in each device, and for this example is negligible. This can be added when there are added more devices in the network, or the devices are going to be identified by name, instead of IP. If you require more information on this, visit the resources at the end of this section.

6. Once the connection is established, log in via SSH to the NVIDIA Jetson. Now you can have access to the NVIDIA Jetson remotely.

WIRELESS CONNECTION TO THE ROS MASTER

Once the hotspot is setup, the user can control the Puzzle-Bot remotely by connecting its PC (client) to the hotspot and access the NVIDIA Jetson through SSH. The command to connect is "`ssh username@IP_address`". For example:

```
ssh mcr2@10.42.0.1
```

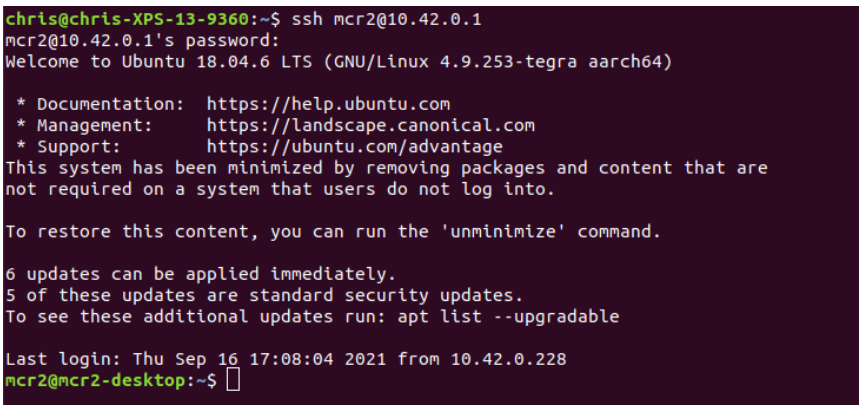


FIGURE 123 SSH CONECTION TO THE NVIDIA JETSON NANO

The password is the one associated with that useame. Now commands such as *roscore* and the ones shown in section *Puzzle-Bot ROS Serial Connection* can be run remotely, however the topics will be associated with the master running on the Puzzle-Bot.

SETTING UP THE USER PC

The ROS running on the user side PC, as shown in Figure 97,must be connected to the master on the Puzzle-Bot for communication to be established.

1. Change the ROS Master URI to match that displayed when `roscore` was started, as an example:

```
export ROS_MASTER_URI=http://10.42.0.1:11311
```

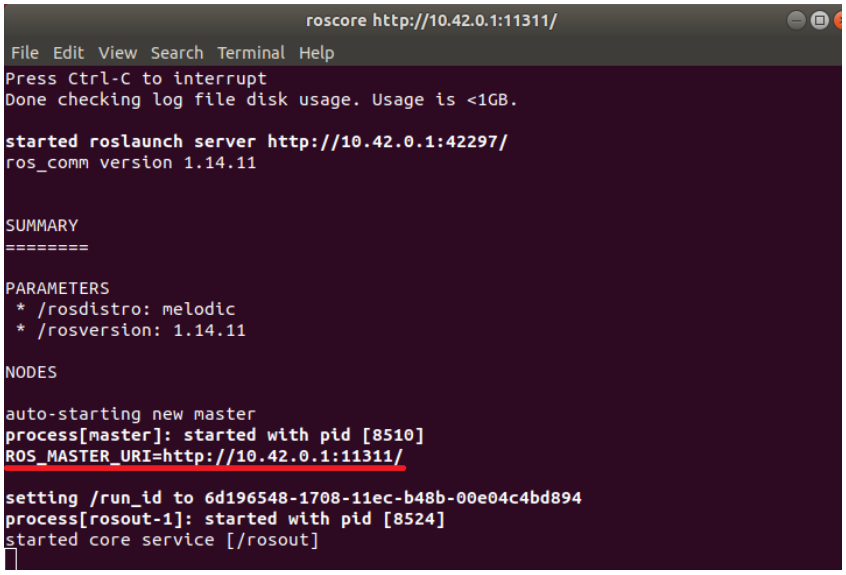


FIGURE 124 ROSCORE RUNNING ON THE EXTERNAL COMPUTING UNIT

2. Next, set the ROS IP and ROS hostname to the IP address of the PC. This can be found by running `ifconfig`.

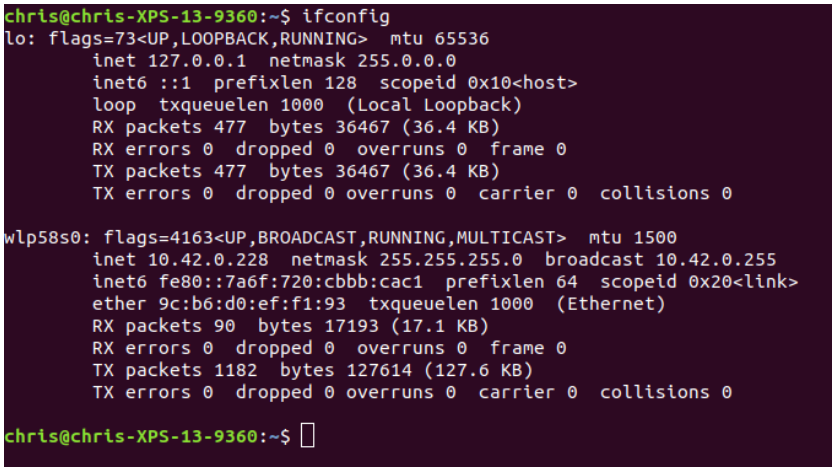


FIGURE 125 IP ADDRESS OF THE CLIENT COMPUTER (PC)

```
export ROS_IP=10.42.0.228
```

```
export ROS_HOSTNAME=10.42.0.228
```

3. These can also be added to the PC's `bashrc` file, as with the Puzzle-Bot. However, if this is done, ROS will not work whenever the Puzzle-Bot is disconnected. It is therefore recommended to set these manually each time the real robot is used.
4. Now, calling `rostopic` should show a list of topics that match those output by the PuzzleBot (see Figure 114). This list will differ depending what mode the PuzzleBot is operating in (see Table 11).

5. The user can now command the Puzzle-Bot to move remotely by publishing to these topics.
6. There are two included examples, both of which use mode 1 (publishing to `cmd_vel`). Similarly, to sections *Gazebo Examples*, `teleop_real_robot` will enable the control of the robot from the keyboard, and `square_real_robot` will use the encoders to drive the robot in a square. These commands are:

```
roslaunch puzzlebot_examples square_real_robot

roslaunch puzzlebot_examples teleop_real_robot
```

   The keybinds for teleop twist operation can be found here:
   http://wiki.ros.org/teleop_twist_keyboard

If you require more information about how to set up ROS network, we recommend the following sources.

http://wiki.ros.org/ROS/NetworkSetup

http://wiki.ros.org/ROS/Tutorials/MultipleMachines

http://wiki.ros.org/ROS/Troubleshooting

PINOUT DIAGRAMS

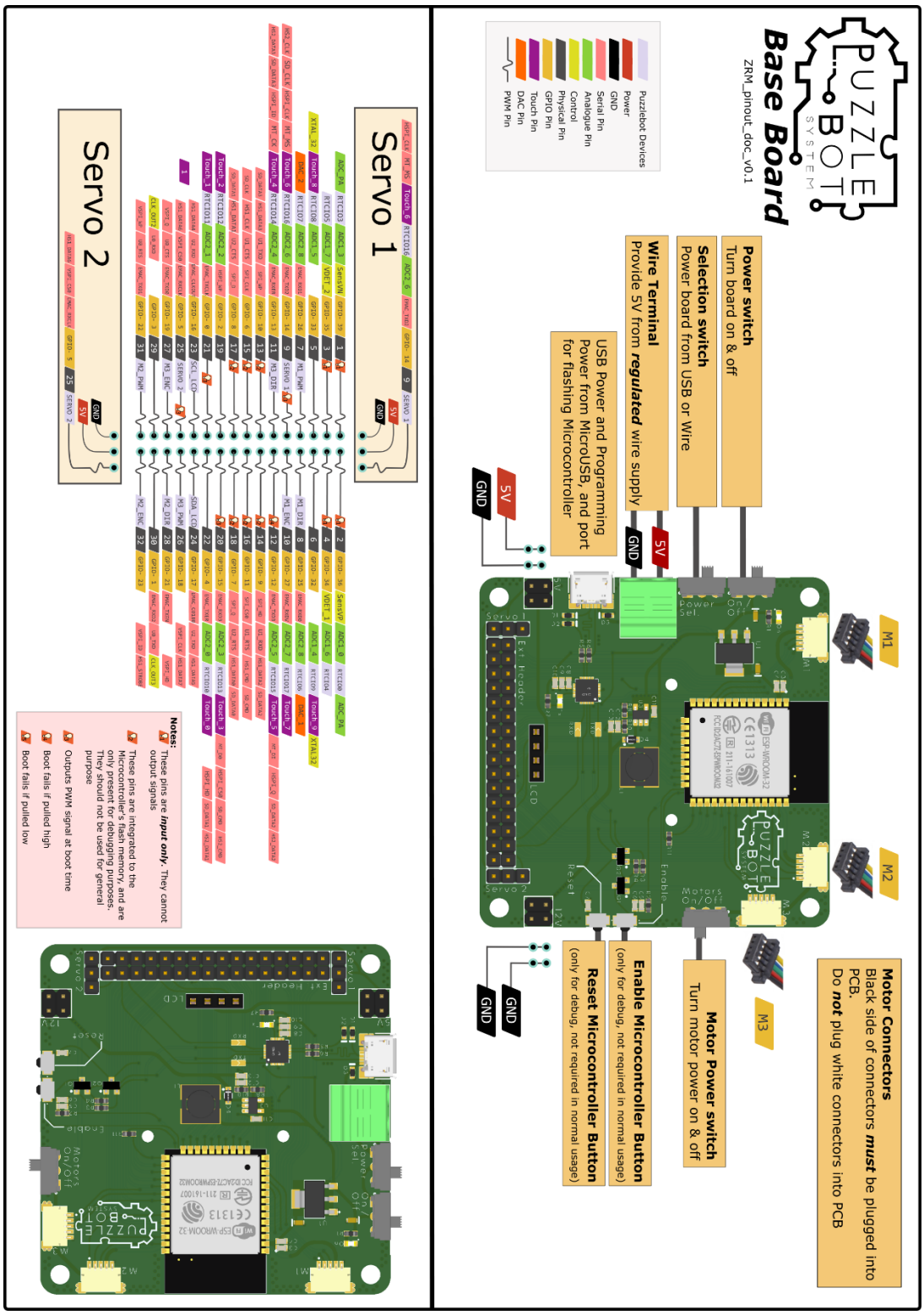
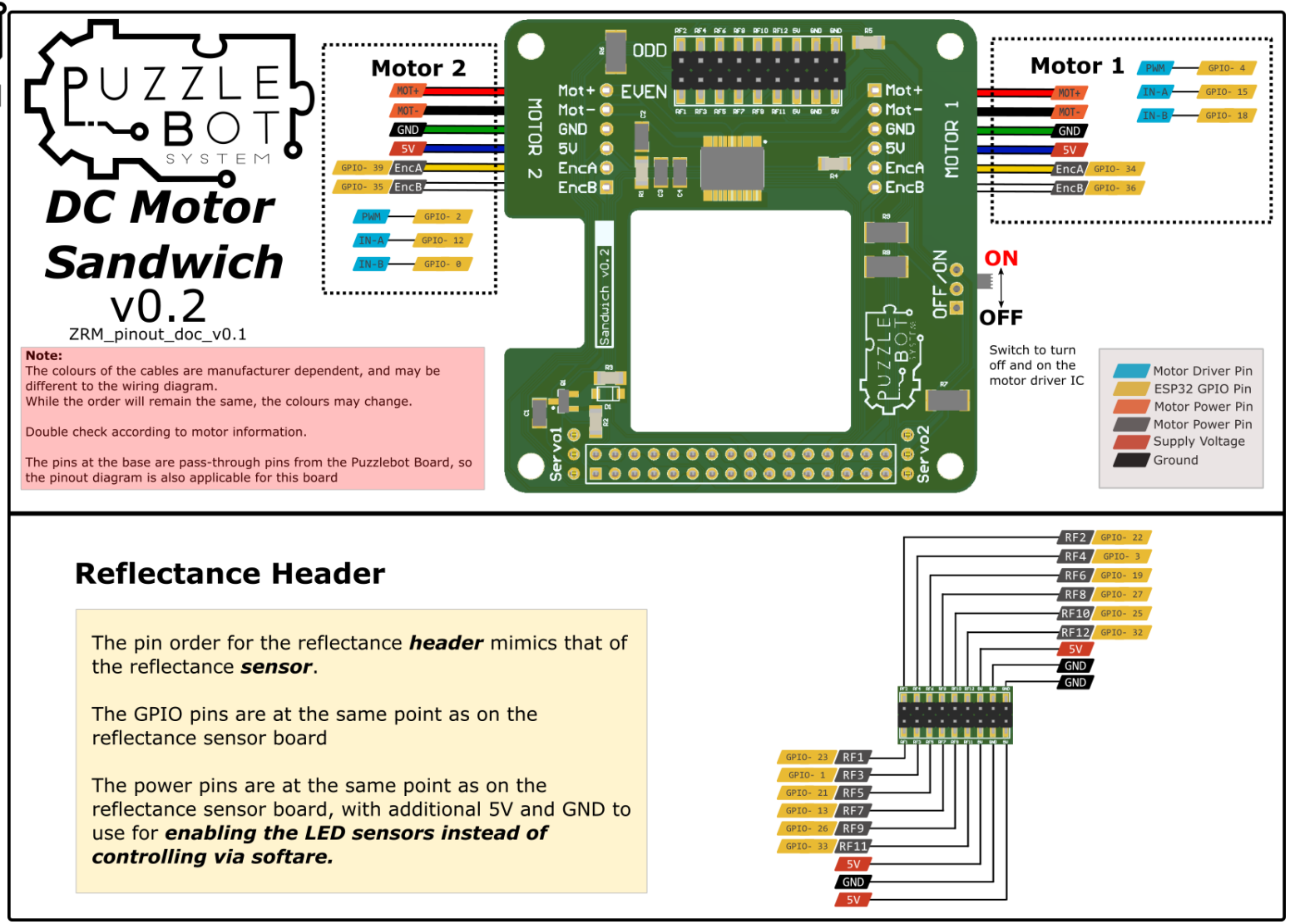


FIGURE 126 PUZZLE-BOT CONTROL MODULE (BOTTOM PCB)



Reflectance Header

The pin order for the reflectance ***header*** mimics that of the reflectance ***sensor***.

The GPIO pins are at the same point as on the reflectance sensor board

The power pins are at the same point as on the reflectance sensor board, with additional 5V and GND to use for ***enabling the LED sensors instead of controlling via softare.***

FIGURE 127 PUZZLE-BOT CONTROL MODULE EXTENSION (TOP PCB)

APPENDIX

SENSOR AND ACTUATOR DATASHEETS

Datasheets for the sensors and actuators can be found in the following webpages.

Motor DC Brushed and encoder:
https://www.pololu.com/product/4824

Motor Brushless and encoder:
https://wiki.dfrobot.com/FIT0441_Brushless_DC_Motor_with_Encoder_12V_159RPM

Servo Motor:
https://www.pololu.com/product/2818

Sonar Sensor:
https://www.parallax.com/product/ping-ultrasonic-distance-sensor/

Reflectance Sensor:
https://www.pololu.com/product/4153

NVIDIA Jetson NANO
https://developer.nvidia.com/embedded/jetson-nano-developer-kit

PUZZLE-BOT PARAMETER FILE

TABLE 12 PUZZLE-BOT PARAMETER FILE

Parameter	Default Value	Description
PidDt:	0.01	Pid controller loop sampling time
Main parameters for the robot		
Robot:		
Type:	1	1-differential drive robot; 2-holonomic robot
ControlInput:	3	1-robot linear and angular velocities; 2-wheel angular velocities setpoints; 3-wheel pwm voltage signals
WheelBase:	0.083	Half of the robot width
WheelRadius:	0.0505	Wheel radius
TopicVx:	VelocityLinearX	Topic for receiving linear velocity of the robot
TopicVy:	VelocityLinearY	Topic for receiving linear velocity of the robot on Y(for holonomic robot)
TopicW:	VelocityAngular	Topic for receiving angular velocity of the robot
Parameters for the right wheel		
RightWheel:		
Motor:		Right motor parameters
Pins:	[4, 15, 18]	Motor driver pins
Sign:	1	Motor direction setting (-1/1)
Type:	2	Motor type. 1-brushless; 2-brushed
Topic:	ControlR	Topic for receiving control pwm
Encoder:		Right encoder parameters
Pins:	[36, 34]	Encoder pins
Sign:	-1	Encoder direction setting (-1/1)
Ticks:	48	Encoder number of ticks for one rotation
Gear:	34	Gear ratio
Type: 2	2	Encoder type. 1-single pulse(no direction); 2-double pulse(with direction)
MeasureType:	1	Encoder velocity measurement type. 1-count pulses; 2-measure pulse duration
Topic:	VelocityEncR	Encoder velocity publish topic
Pid:		Right Pid controller parameters
Kp:	0.1	Proportional gain
Ti:	0.05	Integration time
Td:	0	Derivation time

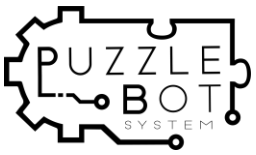
DeadZone:	0.1	Motor control dead zone
Topic:	VelocitySetR	Topic for receiving velocity setpoint
Parameters for the left wheel		
LeftWheel:		
Motor:		Left motor parameters
Pins:	[2, 12, 0]	Motor driver pins
Sign:	-1	Motor direction setting (-1/1)
Type:	2	Motor type. 1-brushless; 2-brushed
Topic:	ControlL	Topic for receiving control pwm
Encoder:		Left encoder parameters
Pins:	[39, 35]	Encoder pins
Sign:	1	Encoder direction setting (-1/1)
Ticks:	48	Encoder number of ticks for one rotation
Gear:	34	Gear ratio
Type:	2	Encoder type. 1-single pulse(no direction); 2-double pulse(with direction)
MeasureType:	1	Encoder velocity measurement type. 1-count pulses; 2-measure pulse duration
Topic:	VelocityEncL	Encoder velocity publish topic
Pid:		Left Pid controller parameters
Kp:	0.1	Proportional gain
Ti:	0.05	Integration time
Td:	0	Derivation time
DeadZone:	0.1	Motor control dead zone
Topic:	VelocitySetL	Topic for receiving velocity setpoint
Parameters for sonar		
Sonar:		
Pins:	14	Sonar pins
Topic:	SonarDistance	Topic for publishing sonar measured distance
Parameters for servo motor		
Servo:		
Pins:	5	Servo motor pins
Offset:	0	Servo motor center offset
Topic:	ServoAngle	Topic for receiving servo motor angle (degrees)
Parameters for reflectance sensor		
Reflectance:		

NPins:	6	Number of pins (number of light sensors used)
Pins:	[32, 25, 27, 19, 23, 22]	Reflectance pins
EmitterPins:	[13, 12]	Emitter pins (for selecting even/odd lights on/off)
EmitterSelect:	2	Emitter selector. 1-even; 2-odd; 3-all
MinValues:	[600, 600, 600, 600, 600, 600]	Minimal reflectance values measured during calibration (use webpage)
MinAvg:	2000	Minimal reflectance average value measured during calibration
Threshold:	1.1	Line detection treshold
Topic:	LineSensor	Topic for publishing line position relative to the sensor
Network parameters		
Network:		
SSID:	Puzzlebot1	Access point ssid
Password:	Puzzlebot72	Access point password
IP:	[192, 168, 1, 1]	Access point IP
RosIP:	[192, 168, 1, 2]	Ros master IP (when ros connection is active)
PortWeb:	80	Port for the web server
PortData:	3141	Port for data communication server (tcp/udp)
WiFiPower:	78	Wifi transmitter power (78 is maximum)
WebPage:	puzzlebot	Domain name for the webpage
Active modules settings		
ActiveModules:		
Servo:	1	Servo motor on/off (1/0)
Sonar:	1	Sonar on/off (1/0)
Reflect:	1	Reflectance sensorr on/off (1/0)
Screen:	1	Screen on/off (1/0)
Ros:	0	Ros communication on/off (1/0)

BIBLIOGRAPHY

[1] R. Siegwart, *Introduction to autonomous mobile robots*, 2nd ed. / Roland ... Cambridge, Mass: MIT Press, 2011.

[2] R. D. Ahmad Abu Hatab, "Dynamic Modelling of Differential-Drive Mobile Robots using Lagrange and Newton-Euler Methodologies: A Unified Framework," *Adv. Robot. Autom.*, vol. 02, no. 02, 2013, doi: 10.4172/2168-9695.1000107.



NOTES