System Design Project Group 16 : User Guide

Raihan Mohammad

Kamen Brestnichki

Diana Kessler

Giuseppe Li

Teodor Todorov

Barbora Murinová

Evaldas Snukiskis

1 Introduction

The following guide gives a detailed description on how to operate our robot. It specifies the libraries that need to be installed, along with their sources, and provides basic troubleshooting steps. The reader should be aware that our system involves components written in Python, Java and C++.

The first section of this document walks the user through the installation process. It is followed by a section, describing the functionality of each component in the system. It also describes how to debug each separate part. The third section gives an overview of the entire system, as well as describe what steps should be taken during match day. Lastly, a troubleshooting guide is provided.

2 Installation

2.1 DICE setup

First, clone the repository by entering the following command into the terminal:

git clone https://github.com/evusas/sdp-group-16

The software has been mostly implemented in Python 2.7, which is installed on DICE and on most Linux distributions. In case Python is not installed, it can be downloaded from www.python.org/downloads/. The only non-standard library which has been used is pySerial. This can be installed using the "**pip**" tool, by typing the following command on the terminal of a DICE/Linux machine:

pip install pyserial –user

Setting up the Python modules required to run the project involves running the "**setup.sh**" shell script located at the root of the project repository. The script itself installs the required Python dependencies using the "**pip**" package management system on your DICE user space. "**setup.sh**" need only be run once.

The vision functionalities in our system requires the OpenCV library. If the library is not already present on your machine, you can install it by downloading it from: https://sourceforge.net/projects/opencvlibrary/

In order to set up OpenCV, run the script "**setup-openCV.sh**", which is located in the java_vision folder.

2.2 Arduino setup

We use the Arduino IDE, which comes pre-installed on DICE computers, to program the Arduino. If unavailable, the Arduino IDE can be downloaded from https://www.arduino.cc/en/Main/Software

To upload code from the computer to the Arduino UNO, a USB to micro USB cable is needed. Connect the Arduino to the DICE machine with the cable and open the IDE. Once the IDE is open, navigate to Tools→Port and ensure that the Arduino is connected to the correct port. To test the Arduino, navigate to File→Examples→Basics→Blink. This opens a sample code that, when uploaded to the Arduino, makes the board's LED turn on and off repeatedly. To verify that the program compiles successfully, click on the top left corner icon "**Verify**". Press the "**Upload**" button, which is situated next to the "**Verify**" icon. The IDE will check for errors and display them in the console, if there are any. The upload sequence is completed when the message "**Done uploading**" is displayed.

Once the Arduino has been tested, the Arduino firmware for our robot can be accessed through Arduino IDE→File→Open→Repository/Arduino/instructioncc/instructioncc.ino - this opens the Arduino sketch file, which then opens all C++ files. Use the upload procedure described above to upload the code to the Arduino. For debugging purposes, we use the serial monitor tool, which can be accessed through the icon located on the top right, that displays any debugging messages included in the code.

For the Arduino and DICE machine to communicate by radio frequency, some setup is required. Connect the SRF stick to one of the machine's USB ports, then follow the instructions found at this link (these instructions can also be found in the Appendix of this document): https://gist.github.com/JFriel/56f58d6fe7a4329c2a2f02dec19a83b

3 System Specification

3.1 Running the entire system

This section describes how all the components below fit together. The entire system is run in two steps:

1) The Java application is run by issuing the " ***./start-system.sh***" command in the java_vision folder. Once that is run, several windows pop up - those are used for vision, as described in section 3.4. After the appropriate vision settings are set, the word "behave" should be typed into the console to start the Strategy module.
2) In a separate console window, set up for execution by following the steps in **Sections 3.2**.
3) Execute ***strategy_controller.run()***. That's it - after a brief delay, your robot should start playing football!

3.2 Setting up Python-Arduino Communications

Please ensure that you have successfully installed the Python dependencies.
The system supports the use of multiple SRF sticks, in case a user wants to control multiple robots with a single machine. Open a terminal in the root directory of the project and execute the following command:

python -i start.py left/right

This script does the following:

- Runs Python in **Read–Eval–Print Loop(REPL)** mode that provides an interactive shell, from which commands can be issued
- Starts the communication bridge between Python and the Arduino
- Gives access to the *Planner* and *StrategyController* instances
- Starts the Java messaging client (World Model sharing)
- Starts the Java messaging server (Strategy listener)

Plug in your SRF stick(s) and switch the Arduino on. The command above will open the communications interface (see Figure 3.3.1) and will prompt the user with the following question "***is it your SRF stick on /dev/ttyACM0 (y/n)?***". Confirm that the recognised SRF stick is the right one by pressing "y"; if "*n*" is pressed, then it will issue the same prompt, but this time with port ***ACM1***. If you have multiple SRF sticks connected, the prompt will cycle through all the possible ports. If the question does not appear when the script is first run, repeatedly press "*Enter*" until it shows up.

Before proceeding, ensure that the connection with the Arduino is established.

3.3 Strategy Controller

The strategy_controller module handles the behaviour of the robot based on the behavioural states issued by the Java Strategy module (shared with Group 15). Each of these states can be tested by following these steps:

1) Follow steps 1) and 2) as described in Section 3.1, omitting the input of “behave” in the Java terminal.
2) The command used to test the strategy module is “***strategy_controller.[state]_strategy()***”, where the [state] can take on any one of the following values: [idle, defend, attack, safe]. Should the user wish to stop the current behavior, they can issue the command ***strategy_controller.stop()***

3.4 Planning

The planning module contains high-level functions, which are used when defining the robot’s behavior. In our codebase, strategy and planning are defined as follows: strategy analyses the world model, decides which state the robot is in (e.g. defend), then calls the necessary functions from the planning module. As such, the planning functions are to be accessed only from strategy. However, these functions can be manually tested by following these steps:

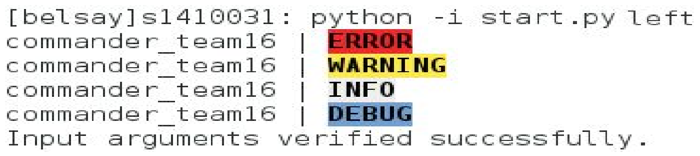
1) Follow steps 1) and 2) as described in Section 3.1, omitting the input of “behave” in the Java terminal.
2) The command used to test the planning module is: ***planner.[function]***, where the [function] argument is the name of the function to be tested.

The program accepts the following input functions:

Command	**Description**
defend_gate() (See Figure 3.4.2)	Moves to the closest point on the trajectory between the gate and the ball and executes shaking movements (see next entry).
shake(shake_num, middle_point, heading_vector) ● shake_num: integer ● middle_point: Point ● heading_vector: Point	This function will move the robot to the left and right of the middle_point for the number of times specified by shake_num. The third argument represents the desired heading of the robot during this maneuver (as discussed in Conventions). This function’s purpose is to trick our opponent’s vision when they’re attempting to shoot for our goal.
go_spot(spot_id, target_heading) (Refer to Figure 3.4.3) ● spot_id: String ● target_heading: integer in range [0, 360], **optional**	The first argument can be any predefined point - currently [“center”, “left”, “right”, “left_safe”, “right_safe”]. If the second argument is omitted, then the robot will turn to the default heading.
pass_xy(target_point): ● target_point: Point	This function assumes the robot has the ball. The robot turns to the point given as an input and shoots the ball in that direction.
get_ball()	Extracts the latest information about the robot and the ball from the WorldModel. It

	then sends the robot to a point that is a set distance from the ball - this is done in order to align the robot for grabbing.

Please refer to the *Conventions section* - in Appendix- for further details on testing the planning module.



```
[belsay]s1410031: python -i start.py left
commander_team16 | ERROR
commander_team16 | WARNING
commander_team16 | INFO
commander_team16 | DEBUG
Input arguments verified successfully.
```

Figure 3.4.1.

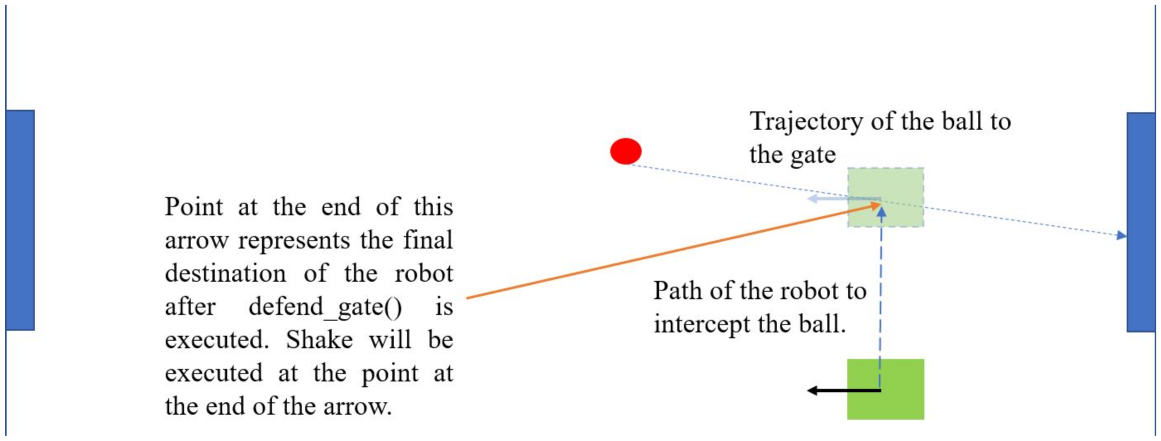

Trajectory of the ball to the gate
Point at the end of this arrow represents the final destination of the robot after defend_gate() is executed. Shake will be executed at the point at the end of the arrow.
Path of the robot to intercept the ball.

Figure 3.4.2

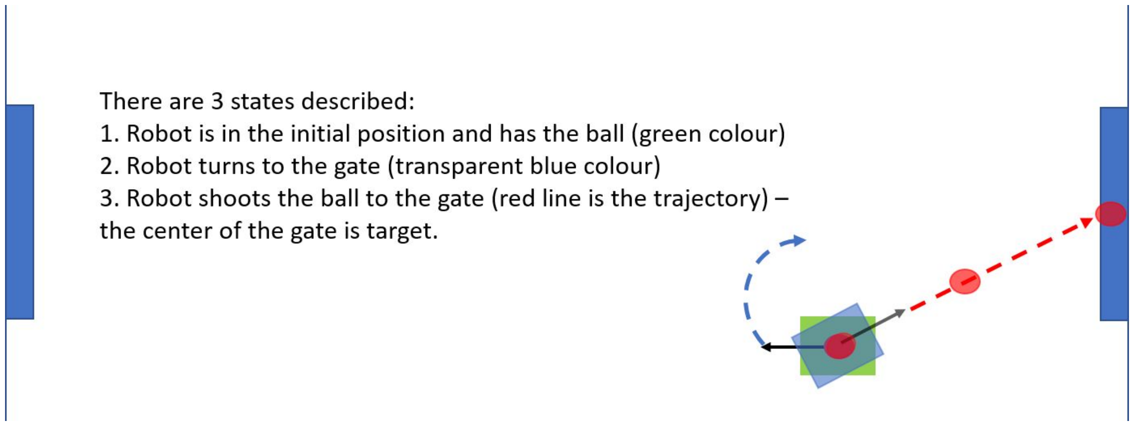

There are 3 states described:
1. Robot is in the initial position and has the ball (green colour)
2. Robot turns to the gate (transparent blue colour)
3. Robot shoots the ball to the gate (red line is the trajectory) – the center of the gate is target.

Figure 3.4.3

Picture above -Figure 3.4.3- illustrates the shoot function- shoot(utils.Point(utils.spots[“right”])). The pass_xy function works in a similar way, but the robot turns to the teammate instead of turning to the gate.

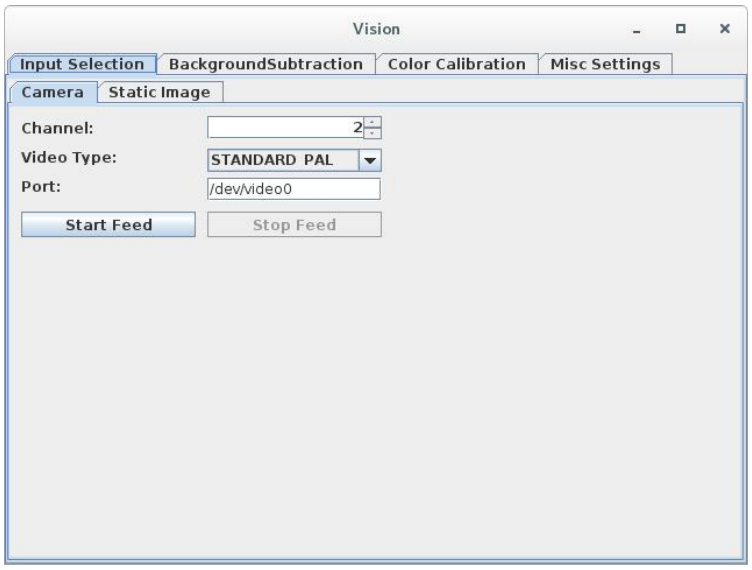
3.5 Vision and Calibration

To launch the vision browse to root/java-vision in a terminal.

Run the "xawtv-setup.sh" to automatically set up the calibrations of the camera itself. If you want to make further changes to the settings type "xawtv" in the terminal and press enter, ignore the error, right-click on the window that opens and that will open the settings for the camera.

To run the vision itself type the "start_system.sh [pitch number]" command, where the last argument should be “1” or “2”, for the pitches in rooms 3.D03 and 3.D04 respectively.

After running the start_system an "SDP Console", "Robot Preview" and "Vision" windows pop up. Click Vision→Input selection→Start Feed. The window should look like Figure 3.5.1.



Channel: 2
Video Type: STANDARD PAL
Port: /dev/video0

Figure 3.5.1. Vision window

That starts the feed from the camera and opens 2 more windows "Strategy" and "Preview". In Preview you can see what the vision sees, as represented by Figure 3.5.2.

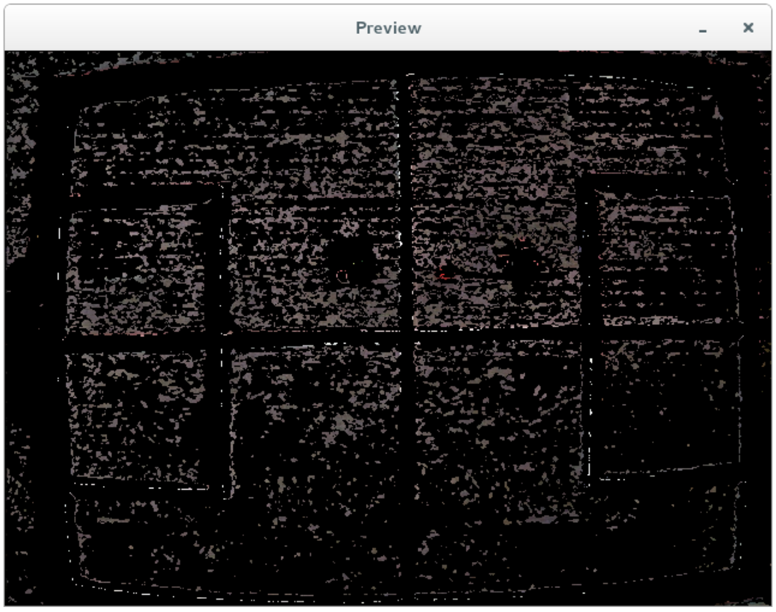
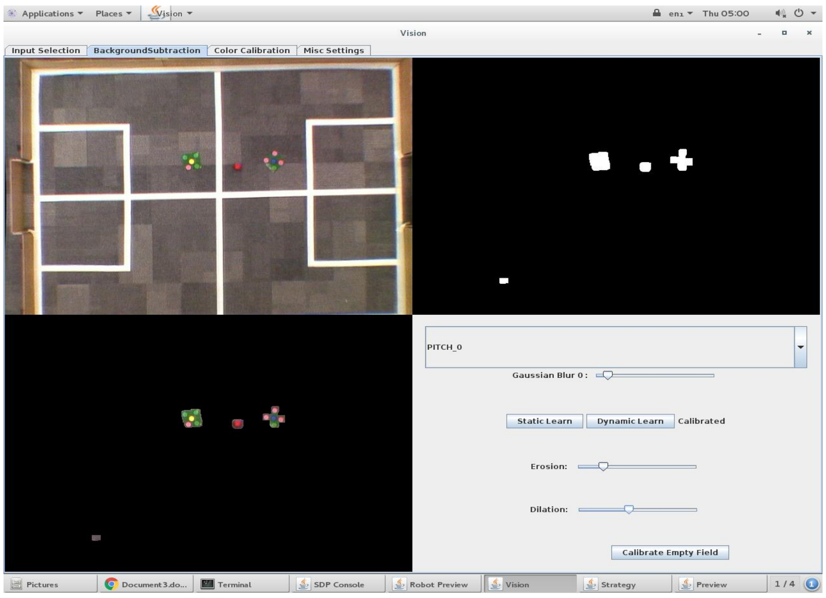


Figure 3.5.2. Preview window before calibration

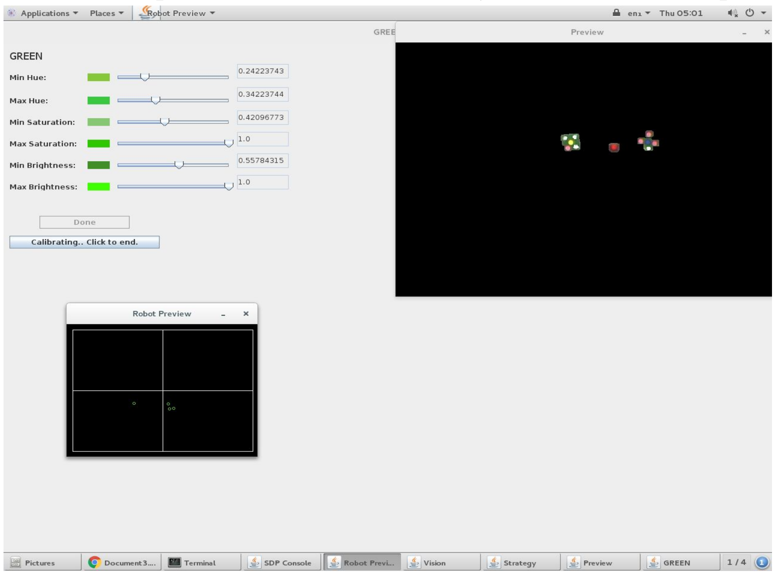
To calibrate the vision, maximize the Vision window screen, go to the "Background Subtraction" tab, as shown in Figure 3.5.3 and after entirely clearing the pitch, press the "Dynamic Learn" button to automatically calibrate the background subtraction. This will take about 10 seconds to calibrate.



PITCH_0
Gaussian Blur 0 :
Calibrated
Erosion:
Dilation:

Figure 3.5.3. Background Subtraction tab after successful calibration

Add the plates and the ball to the pitch and go to the "Colour Calibration" tab in the Vision window, then press on any of the colours and press the "Calibrate" button.
A new window for the colour opens up. If you press the "Calibrate" button on the new window, it should look like Figure 3.5.4 and there you can click on the desired colour in the "Preview" window to set it automatically. You can further change the calibrations by adjusting the sliders to clear up any noise or add points that were excluded by the automatic set up.



GREEN
Min Hue: 0.24223743
Max Hue: 0.34223744
Min Saturation: 0.42096773
Max Saturation: 1.0
Min Brightness: 0.55784315
Max Brightness: 1.0

Figure 3.5.4. Example of calibration for GREEN

The last step is to produce calibrations you are satisfied with for all the colours. At this point all the important points should be clearly visible in the "Robot Preview" window and all the robots should be named. The “Robot Preview” window in Figure 3.5.5 is given for reference.

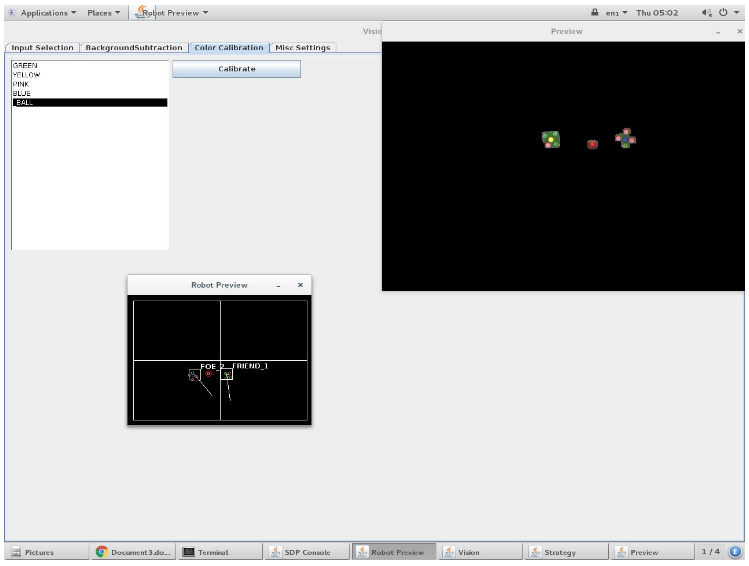


Figure 3.5.5. End result after the all the calibrations are done

You can also go to the "Misc Settings" in the Vision window to save the current calibrations or assign roles to the robots [FRIEND_1, FRIEND_2, FOE_1, FOE_2].

3.6 Structural Information

3.6.1 Movement

Four NXT motors, and 4 holonomic wheels arranged in an “X” shape (with respect to the horizontal axis of the robot), control the robot’s movements. This allows the robot to move in four directions: forwards, backwards, and sideways.

3.6.2 Kicker

The Kicker is a solenoid, which has a separate battery placed at the back of the robot. This helps the robot deliver a more powerful kick. Since the solenoid requires a relatively large amount of power, it is, as such, connected to a relay, which acts as a switch. The connection and the circuitry can be seen in the Figure 3.6.1 and 3.6.2.

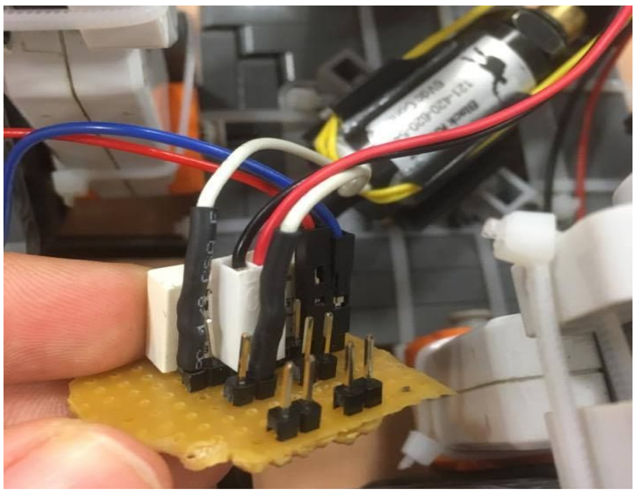


Figure 3.6.1. Solenoid circuitry and the connections

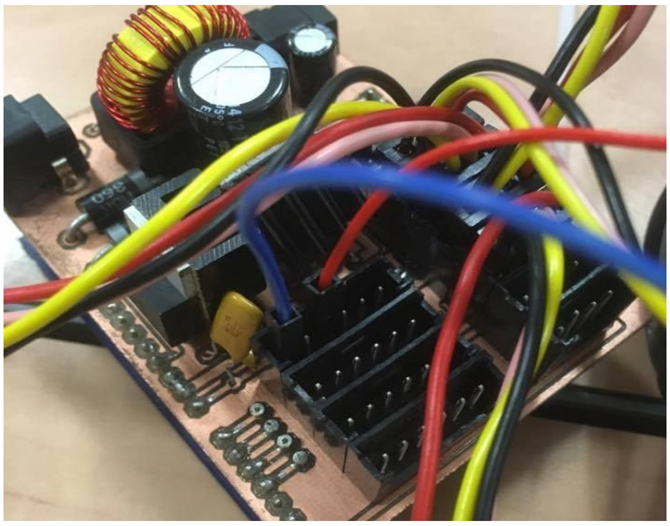


Figure 3.6.2. The blue and red jumper wires connect to the power board

3.6.3 Grabber

The grabber utilizes a small motor, which is directly connected to a Lego piece. Two pegs connected to the robot hold the ball in place. Once a grab command is issued, the robot confirms that the ball has been grabbed with a light sensor placed above the two ball holding pegs. Figure 3.6.3 shows the two holding pegs and sensors. Figure 3.6.4 shows the grabber.

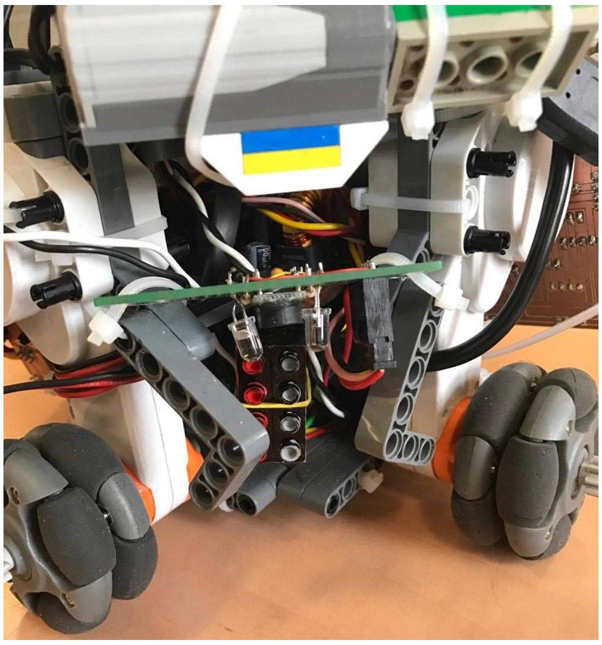


Figure 3.6.3. The 2 grabbing pegs and the light sensor

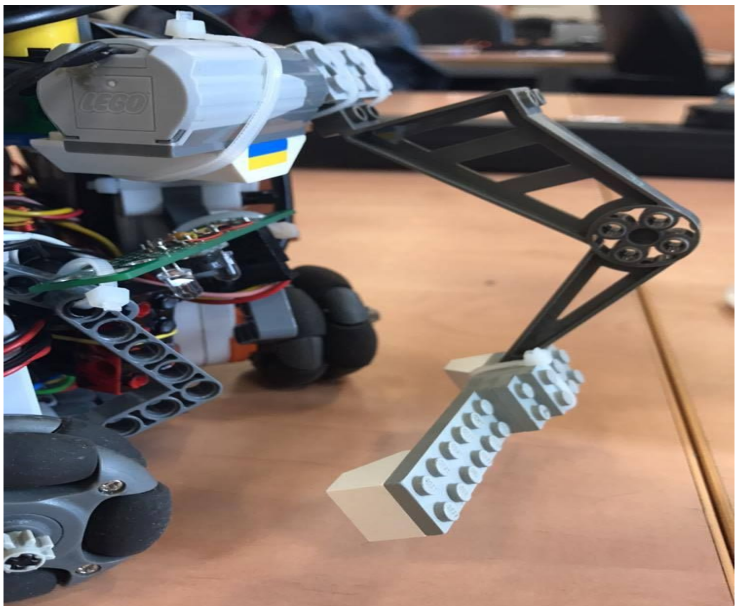


Figure 3.6.4. The Grabber along with the motor.

5 Troubleshooting guide

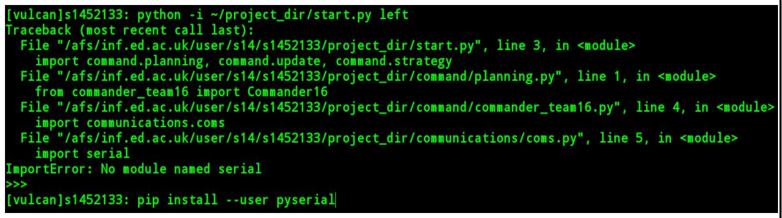
The following table lists common faults in the system and how they can be tackled.

5.1 Arduino and Solenoid

Fault	Solution
The Arduino loses power.	The Arduino might have reset itself because it has crashed. Check the firmware.
The robot struggles to move.	This happens due to low battery voltage. Replace with a fresh battery pack.
A wheel does not spin.	Check the connections (the NXT wire, the NXT to 6-pin splitter, the motor board).
The batteries overheat.	Check for any direct contact between copper as that may cause shortages. This is potentially dangerous to the board!
Not all the wheels are touching the ground.	Make sure the cross on top of the robot is in place.
One wheel is weaker than all the rest.	Check for any item inside the robot which may cause friction on the axle.
The robot cannot perform a kick.	● Make sure the jumper wires used to connect the small circuitry provided for the solenoid to the Arduino are connected (The rightmost one is a bit loose and comes off often). See Figure 3.6.1. ● Check the elastic band which is used for the retraction of the solenoid.

5.2 Communications

Behaviour	**Notes**

 `[vulcan]s1452133: python -i ~/project_dir/start.py left`
`Traceback (most recent call last):`
`  File "/afs/inf.ed.ac.uk/user/s14/s1452133/project_dir/start.py", line 3, in <module>`
`    import command.planning, command.update, command.strategy`
`  File "/afs/inf.ed.ac.uk/user/s14/s1452133/project_dir/command/planning.py", line 1, in <module>`
`    from commander_team16 import Commander16`
`  File "/afs/inf.ed.ac.uk/user/s14/s1452133/project_dir/command/commander_team16.py", line 4, in <module>`
`    import communications.coms`
`  File "/afs/inf.ed.ac.uk/user/s14/s1452133/project_dir/communications/coms.py", line 5, in <module>`
`    import serial`
`ImportError: No module named serial`
`>>>`
`[vulcan]s1452133: pip install --user pyserial`
	Your system seems to be lacking dependencies. Try running "setup.sh" and if it fails install the missing package manually by writing "*pip install --user <package_name>*"

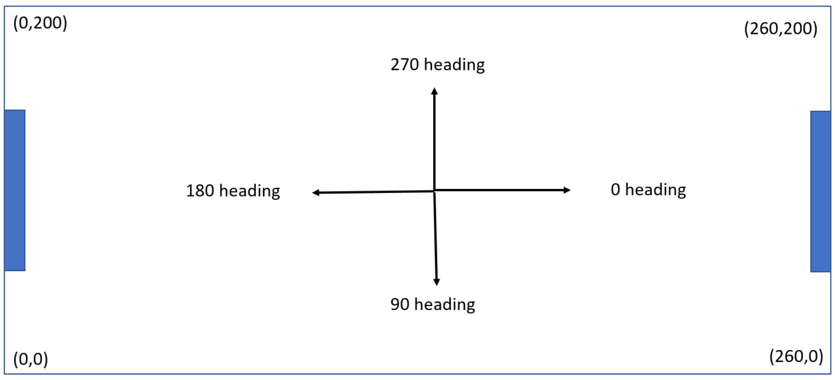
The robot does not respond to commands.	Execute "Coms.reset()" command in REPL.
coms | Trying to connect to SRF ... coms | ### ALL PORTS FAILED - SRF NOT INITIALIZED ###	Reconnect the stick, press [Enter] several times and look for new information.
############# SUCCESS ############## ### /dev/ttyACM0 SERIAL ENABLED ###	SRF stick connected successfully.
coms | Arduino just reset	Computer-Arduino communication established.

APPENDIX

1 Conventions

1. The (0,0) point is in the lower-left corner of the pitch.
2. The size of the pitch takes values from 0 to 260 on x-axis and from 0 to 200 on y-axis.
3. The 0 heading is parallel to the x-axis and points to the right gate. The heading of the robot increases clockwise.

(See the picture below for a sketch that explains the conventions.)



4. For testing planning: for giving a Point as an input, please write utils.Point(x_coordinate, y_coordinate), where the value of x_coordinate varies from 0 to

260 and the value of y_coordinate from 0 to 200.

2 SRF Setup

The port number can be found by connecting the Arduino to the port and following the following steps to find the port as talked about previously.
The configuration of the SRF stick can be changed by entering the command mode. To enter the command mode press +++ (You won't be able to see what you type). If you successfully enter the command mode OK is displayed. You only have 5 seconds between two commands in the command mode before the system exits it, so make sure to type fast. If the command entered is wrong the ERR is displayed and OK otherwise. Once in command mode press Ctrl-a Shift-c to clear the output from the previous command. Execute the following commands pressing ctrl-a shift-c after each command

- ATRE
- ATAC
- ATWR
- ATDN

The following commands set the SRF stick to factory settings and you automatically exit the command mode. Therefore enter the command mode again by pressing +++ and execute the following commands.

- ATID00XX (XX being your group number)
- ATAC
- ATRP1
- ATAC
- ATCNXX (XX denoting the group frequency number)
- ATAC
- ATWR
- ATDN

The SRF stick has been configured for the group settings after execution of the above commands so exit the screen by pressing ctrl-a k and unplug it. Plug the Arduino in the same port and enter the screen using the $ screen PORT command (where PORT is the port number). Reset the Arduino radio device by executing the following commands after entering the command mode
.

- ATRE
- ATAC
- ATWR
- ATDN

Once the Arduino has been reset to the factory settings it can be configured to the group settings. Enter the command mode and execute the following commands.

- ATID00XX (XX denoting the group number)
- ATAC
- ATRP1
- ATAC
- ATCNXX (where XX is the group frequency)
- ATAC
- ATBD 1C200
- ATAC

- ATWR
- ATDN