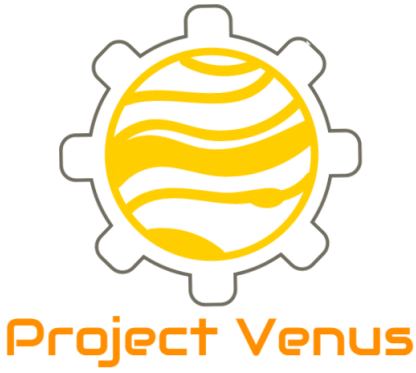

User Guide

SDP Group 15-H

Emilia Bogdanova
Patrick Green
Julijonas Kikutis
David McArthur
Aseem Narang
Ankit Sonkar

The University of Edinburgh

Contents

1 Installation

1.1 Cloning the repository and setting up

To clone the repository, execute in terminal:

```
$ git clone https://github.com/julijonas/venus.git
```

Then go to the root directory of the project and create a Python virtual environment which will contain a local copy of the libraries needed for this project:

```
$ cd venus
$ virtualenv --system-site-packages env
$ source env/bin/activate
```

Install the *pyserial* library which is used to send data to Arduino through RF stick:

```
$ pip install pyserial
```

To set the persistent radio chip configuration for a new Arduino board, connect to the PC with an USB cable, execute `screen /dev/tty0` in terminal, and enter the following commands substituting the appropriate group number and radio frequency band:

```
+++
ATID0015
ATAC
ATRP1
ATAC
ATCN80
ATAC
ATWR
ARDN
```

To close `screen`, press *Ctrl+A* and then *X*. The identical steps have to be performed for a new RF stick.

To program the Arduino to receive and execute messages, open the Arduino IDE by executing `arduino` in terminal. You'll need to add three libraries which can be found under `arduino/` in the project directory: *ArduinoSerialCommand*, *SDPArduino*, *SimpleTimer*. In order to add a library, go to `Sketch -> Import Library... -> Add Library...` and choose the library directory you want to import. Then open the Arduino file `arduino/arduino.ino` using `File -> Open...` and upload it to the board using `File -> Upload`.

2 Hardware overview

2.1 Motors

The wheels are powered by NXT motors on a gear system of 2 : 1. A reduction of power of one wheel can be caused by the gears slipping out of line. The driving motors

are labelled 0 to 3 from the back right clockwise. Simple motions can be tested using the `move direction_angle turn_angle` command where `direction_angle` defines the movement vector and `turning_angle` defines the rotation whilst moving on that vector. The gears in the grabber can come out of line as well when crashing so ensure that when open the enter arm span occurs. The grab has a motor on one arm and uses a gear ratio of $25 : 10 : 10 : 25$ to ensure each arm is opened at the same rate and in the correct direction. A mini motor is used for this and therefore the actual motion is calibrated using a time step and not encoders. Please see the Technical Specification for an explanation of the kick. As it relies on timing, the open grabber with a spin jammed grabber arms can reduce accuracy.

2.2 Sensors

Each NXT motor is connected to the rotary encoder board see the specification for an image. The connections in the anticlockwise order from the top are: the I2C bus to the Arduino, back right motor, back left motor, front left motor, and front right motor. Using this board the information about the amount of rotations the motor has performed since the last query is available for the Arduino code as a separate integer for every motor. Every 5 ms the board is queried whether the target value has been reached. After the average of the rotary values of all four motors becomes greater or equal to the target value, the motors are stopped. If plugged in incorrectly motors will run continuously regardless of encoder value.

The light sensor is located above the grabber. The sensor returns a value associated with the reflected colors in its line of sight. Then the Arduino compares that value to a predefined threshold corresponding to the red ball. Sometimes a white line inside the pitch can be mistaken for the ball noticeable in the games as a random kick whilst in a state of grabbing. Ensure `query_ball` is working before playing and adjust the threshold as outline in calibrations.

3 Usage

3.1 Preparation for the match

In order to turn the robot on, connect the battery pack to the power board. The pack consists of 10 AA rechargeable batteries. Normally one fully charged battery pack lasts for 7-10 minutes after which a noticeable under-turning will occur. Ensure that the battery back is placed directly in the center as a change in weight distribution can require a recalibration of moving, turning, and kicking.

To communicate with Venus, the RF stick should be connected to the machine you are running the commands from. Then after performing steps described in Section 4.1, you will be able to operate the robot.

If you make any changes in the Arduino code, you need to upload them before running the commands as detailed in Section 1. Uploading can be done either via the RF stick or USB cable connected to the Arduino board.

NB: If another robot is doing the kick-off in a game, do not start the strategy by typing `hs` before that robot performs the kick-off.

3.2 Robot calibration

Before playing a match, ensure that the spin-kick using `goal`, forward motion using `f`, and turning using `c` are calibrated correctly. This can be done by repeatedly executing the respective commands with different input values, changing the data points in the `calibration.ods` spreadsheet, and putting the resultant equations from the spreadsheet to `ee`, `f`, and `c` methods in `control/holonomic.py`.

When calibrating the spin-kick the aim is for the robot not to be fully turned towards the goal before the kick. To achieve that, there are different "correction angles" for the six main zones of the pitch as seen in Figure 1. These angles can be calibrated in `strategy/simple_holonomic.py` in `shot_correction` method.

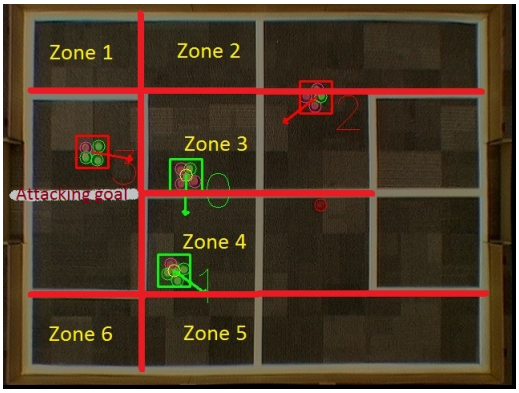

Figure 1: Different shooting zones

The best strategy here would be to place the robot with the ball in different sections of one zone and execute the `goal` command choosing a constant for the optimum orientation that achieved the best result. Another important calibration is checking whether the light sensor threshold correctly identifies the grabbed ball and lack thereof. The threshold is defined in the `query_ball` method in `control/holonomic.py`.

4 Software overview

4.1 Running instructions

Run the following command in the terminal from the project root directory if it has not been done already to enable the Python virtual environment which contains the required libraries:

```
$ source env/bin/activate
```

In order to set the room containing the pitch, colors of the robot, and side of the goal, change the hard-coded parameters of `init` method in `control/holonomic.py` as detailed in Table 1. For example, if the room is 3.D04, the robot has the yellow team color and one green-coloured dot, and the robot is defending the goal closer to the computers, the line would be:

```
def init(self, room_num=1, team_color='yellow', our_color='green',
    computer_goal=True):
```

Room	room_num
3.D03	0
3.D04	1

team_color
'yellow'
'blue'

our_color
'green'
'pink'

computer_goal
True
False

Table 1: Allowed parameters for the `init` method

Then ensure that the RF stick is plugged in. To start the application, make connections to the RF stick, vision feed, and get access to the command prompt, execute the following line:

```
$ python main.py
```

NB: In case you make changes to the Arduino code, you should upload your changes to the board as detailed in Section 1.1.

Then the operator of the robot can start entering the commands. The main command is `hs` which starts the strategy. The listing of all commands is provided in Table 2.

Constantly run the strategy state machine	`hs`
Output a picture of the potential field of the state	`map state_name`
Perform a holonomic motion (angles in degrees)	`move direction_angle turn_angle`
Move forward (in cm, negative means backward)	`f distance`
Rotate clockwise (in degrees, negative means anti-clockwise)	`c angle`
Stop all motors	`s`
Open the grabber	`o`
Close the grabber	`g`
Perform a spin kick	`ee`
Print world state	`w`
Query light sensor	`query_ball`
Pass the ball to the teammate	`pass_ball`
Catch the ball coming from the teammate	`catch_ball`
Kick the ball to the goal	`goal`
Exit the application	`exit`

Table 2: Commands available for the operator of the robot

The `goal` and `pass_ball` commands perform shooting. First, the robot turns so that the optimum orientation is met. Then, an additional grab is done followed by a kick initiated through the `ee` command. Sleep is added before the kick to reduce the irregularities in motor powers caused by the previous movement. The `catch_ball` command moves the robot to face its team mate. A grab is initiated once vision detects that the ball is within a range of 32 cm set exactly for vision feed speed dependent on camera. If the ball never enters this range and stops moving, vision detects this which exits the command. The `ee` command instigates a kick by opening the grabber and spinning for an encoder value of 200, in that order. Irregularities in accuracy can come from both running over white tape on the pitch and also battery level.

4.2 Calibration of the vision

Camera capture settings: As the images appearance can change between computers you may need to slightly adjust the capture settings. This simple calibration is always more preferable than going for a complete calibration. In which case use the slider bars that pop up on the vision feed each corresponding to a percentage as seen in Figure 2. Brightness and contrast can help make the colors more distinguishable and a larger contrast than brightness works better. Reducing the saturation can help remove unwanted spots from the blurred colors created by the white lines.

Color calibration: To enter the complete calibration mode the slider bar named calibration must be moved. The first step is to click and the terminal will outline everything that can be calibrated as seen in Figure 3. Each option is activated by pressing a key outlined in the terminal. For colors it is advised to click on that coloured spot a few times. Be sure to check the terminal to make sure your on the right color. For pitch dimensions a description of how to click each object is added in the terminal. The general convention is any order to mark the goals and for shapes start at the top left and work clockwise. Once you ready to move on press the ESC key.

Figure 2: Vision feed window

Fine tuning the colors: Once the ESC key has been pressed the color thresholds predicted from the clicking can be fine tuned in various windows similar to the window in Figure 4. If nothing is visible, it is advised to reduce saturation and value first before adjusting the hue values. The aim is to see as minimal of spots not in that color as possible and make the actual color as solid as possible. Press the ESC key to move on

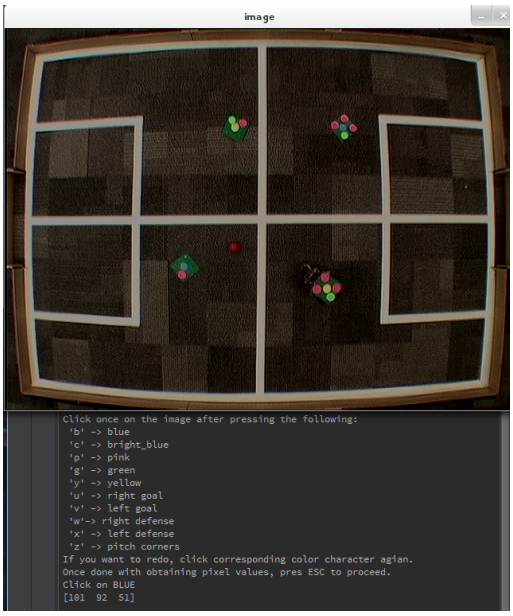

Figure 3: Calibration window

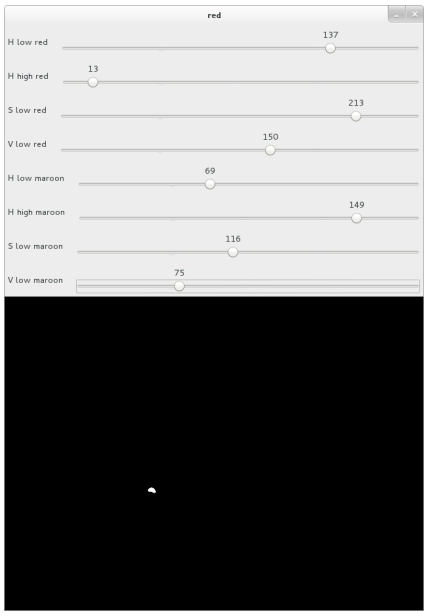

Figure 4: Color tuning window

and eventually the original vision system will be brought up.

Green and yellow: As they can blend together easily it is important that they can be seen clearly so try to reduce the hue threshold of yellow and increase the hue threshold of green as much as possible.

Red and pink: Sometimes the ball cannot be found so if you enter the calibration and press ESC instantly you will only be adjusting reds thresholds. The tuning window for red requires you to adjust to separate ranges of values so make sure this is done to get as clear spot as possible. Its similarities with pink can be avoided after calibrating by adjusting the slider bars for the camera capture settings.

5 Troubleshooting guide

Problem 1: When uploading the code to the Arduino board, the error message appears:

```
processing.app.SerialNotFoundException: Serial port '<port>' not found. Did you
    select the right one from the Tools > Serial Port menu?
```

Solution: Make sure you are not running the Python application at the same time or have not opened the serial interface any other way when the changes are being uploaded. The Arduino IDE is only able to program the Arduino when the serial interface of the RF stick or Arduino itself is registered as `/dev/ttyACM0` on the PC. If there are other programs accessing `/dev/ttyACM0` and the RF stick or USB cable is unplugged and plugged again, the "new" device is issued `/dev/ttyACM1` instead by the Linux kernel.

Problem 2: When sending commands the robot does not respond although it should.

Solution: Power cycle the Arduino board.

Problem 3: The Python application cannot connect to the RF stick because the RF stick has registered itself as `/dev/ttyACM1`.

Solution: Either close all programs that have handles to `/dev/ttyACM0` and reconnect the RF stick or change the `device_no` parameter in the `connect` method in `holonomic.py` to 1 and restart the Python application.

Problem 4: The commands are evidently sent but the robot does not move.

Solution: Power cycle the Arduino. If that does not help, change the battery set to a fully charged one.