System Design Project
2015-2016

Velosso User Guide

Paul Sinclair : s1337523
Dovile Vitonyte : s1237357
Karen McCulloch : s1340952
Krassy Gochev : s1346707
Tomislav Jovanovic : s1341139
James Friel : s1332298

Mentor
Nantas Nardelli

Contents

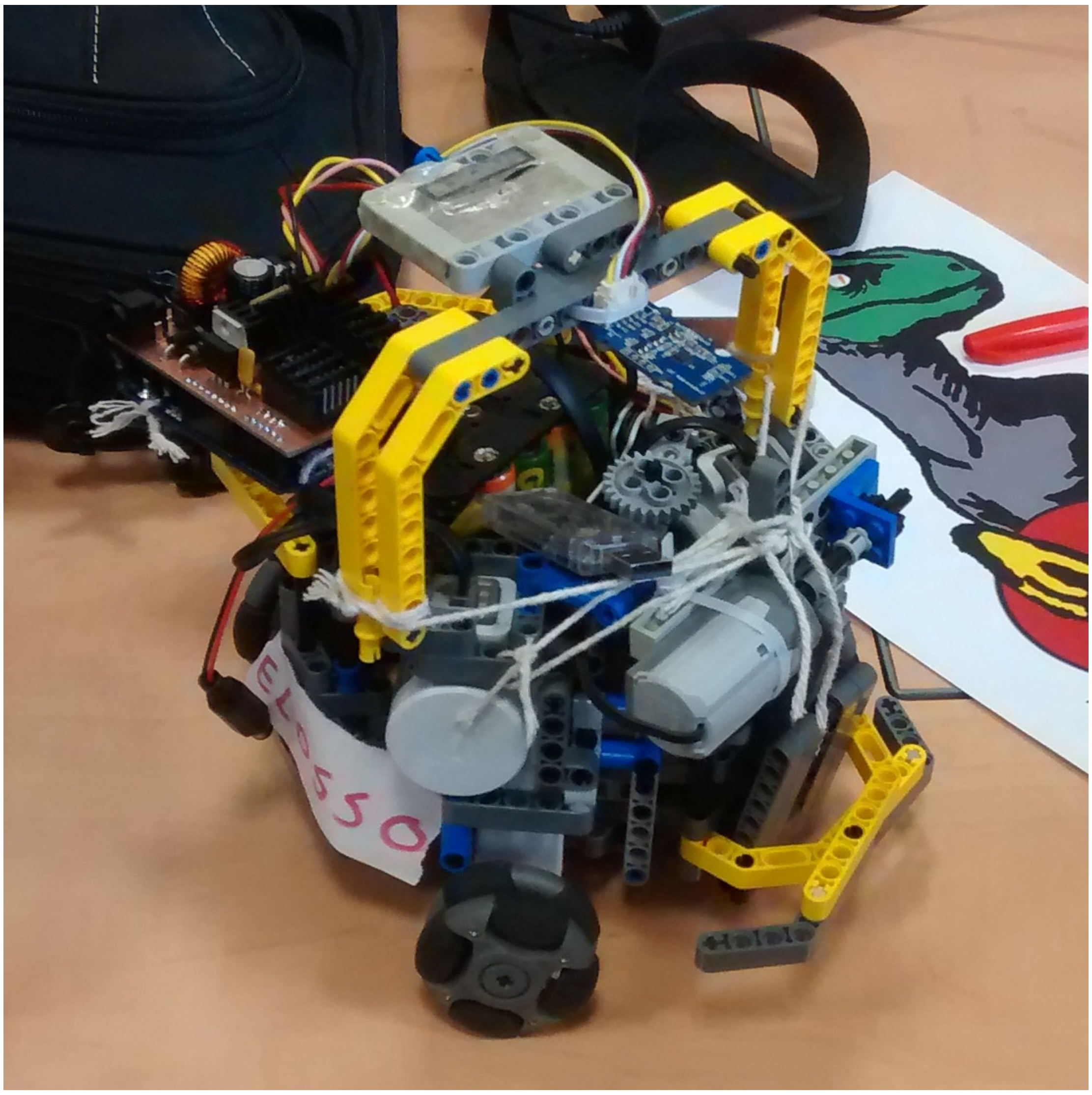


Figure 1: Velosso, The Robot

1 Introduction

The Velosso robot is an versatile robot football player built by team B4 for the 2016 SDP course. The system is configured to operate under the setup of two teams, yellow and blue, with two members in each team and a birds eye vision feed of the pitch.

1.1 System Overview

The robot is a three wheeled Lego robot, using Lego NXT Motors and hobbyist electronics (see figure 1). It is operated over RF by a DICE computer which receives a live video feed of the pitch.

1.2 Hardware

Thanks to its holonomic wheels, the robot can move in any direction and can also turn itself while moving for maximum speed and accuracy. It uses a standard claw-style grabber to catch the ball and bring it in to its kicker, a paddle that can hit the ball at varying powers, allowing it to kick the ball any distance.

1.2.1 On the robot

- 3 holonomic wheels: These allow it to move in any direction equally easily. They're connected to the external of the frame with rotational symmetry.
- 8xAA battery pack: Powers the robot. This lies on the top of the robot, at the center of the frame, to keep the center of gravity low and steady.
- 3 NXT interactive servo motors: Each motor powers one of the holonomic wheels.
- 2 PF Medium motors: One powers the kicker, the other powers both claws of the grabber.
- Rotary encoder board: Tracks the positions of the motors.
- Motor control board: Instructs the motors.
- Compass and accelerometer: Tracks the robot's orientation locally, used for auto-adjustment. We attempted to place these in a Faraday cage to improve their accuracy, but this proved ineffective.
- Arduino xino: Communicates with the rotary encoder board, compass and accelerometer and the motor controller board, and receives and processes commands from the computer.

1.2.2 Accessories

- USB to mini USB cable: used to upload code to the Arduino
- RF stick: used to send commands to the Arduino during a game
- Battery charger with 8 spare batteries. Remember to keep one set of batteries charging at all times.
- Battery tester

1.3 Software

The software is split into 5 parts:

1.3.1 Arduino

The arduino implements a finite state machine for a communications protocol, performing various commands such as moving a certain angle and displacement, grabbing and polling sensors for data. The concurrency based on its master finite state machine allows it to perform the above at the same time.

1.3.2 Vision

The vision system is separate from the other sub-systems and is run on a separate process. It independently observes the pitch and identifies objects by using a camera feed (or a static image) as well as making data available to `planning/world.py` which is used by the planner. The main vision thread can be found at `vision/sender.py`.

1.3.3 Strategy

The strategy is a modular system that models the world based on the information received from the vision feed and constructs and executes plans of action based on its beliefs about the world. Run this by executing `$ cd planning && python robotAI.py`

1.3.4 Simulator

The simulator allows you to test out the strategy and planning system in a closed environment, letting you check that the software side of the system runs as intended before testing it on the robot to check that it works in practice too and tweak as necessary. It models our robot and the ball's movement by generating the expected position, speed, and other properties of both of these at each moment in time, assuming a perfect world. To run the simulator, set the simulator flag to `True` on line 6 of `planning/conf.txt`. This will enable the simulator and disable the vision feed and RF stick. Having done this, simply execute the planning system as normal (`$ cd planning && python robotAI.py`).

1.3.5 Communications

Robot communication works by sending serial commands over RF to the robot. The communication software comes built with simple instruction converters for ease of use.

2 Getting Started

2.1 Installation

The system requires various external libraries and packages, namely an Arduino IDE, xawtv and Python 2.7.5 with numpy and cv2.

2.1.1 DICE installation

To install all the required python modules, run

```
$ pip install -r requirements.txt --user
```

2.1.2 Arduino installation

To load the Arduino code onto the Arduino, launch the Arduino IDE through the terminal (`$ arduino`) and open `arduino/arduino.ino`. Connect the Arduino to the PC using the provided USB to mini-USB cable, then click upload. Ensure the batteries are not connected to the Arduino as this can prevent successful uploading. Remember to plug the batteries back into the robot before running the system though.

2.2 RF stick Setup

The system comes configured to write to serial port ACM0 and the RF stick is set to broadcast on ATCN frequency 0x27. The serial port configuration can be changed in `comms/commsThread.py` and the broadcast frequency is changed by values of the RF stick during setup, details of which can be found at `jamesfriel.uk/sdp_srf_setup`

3 Configuring the System

3.1 Strategy Configuration

The strategy system configuration can be found in the configuration file `planning/conf.txt`. Before each match, ensure that all settings are assigned the appropriate value. Line one is which team the robot on (`yellow` or `bright_blue`), line two is the most common dot colour on our robot (`pink` or `green`), line three is what colour the ball is (`red` or `blue`), line four is which goal is the opponents' (`left` or `right`), line five is whether the simulator is to be used (see § 1.3.4) (`true` or `false`), and line six is which pitch we're on (`pitch0` for 3.D04, or `pitch1` for 3.D03).

The planning system is currently configured to handle a pitch with dimensions of 3m x 2.2m; this can be altered in constants.py.

3.2 Vision Calibration

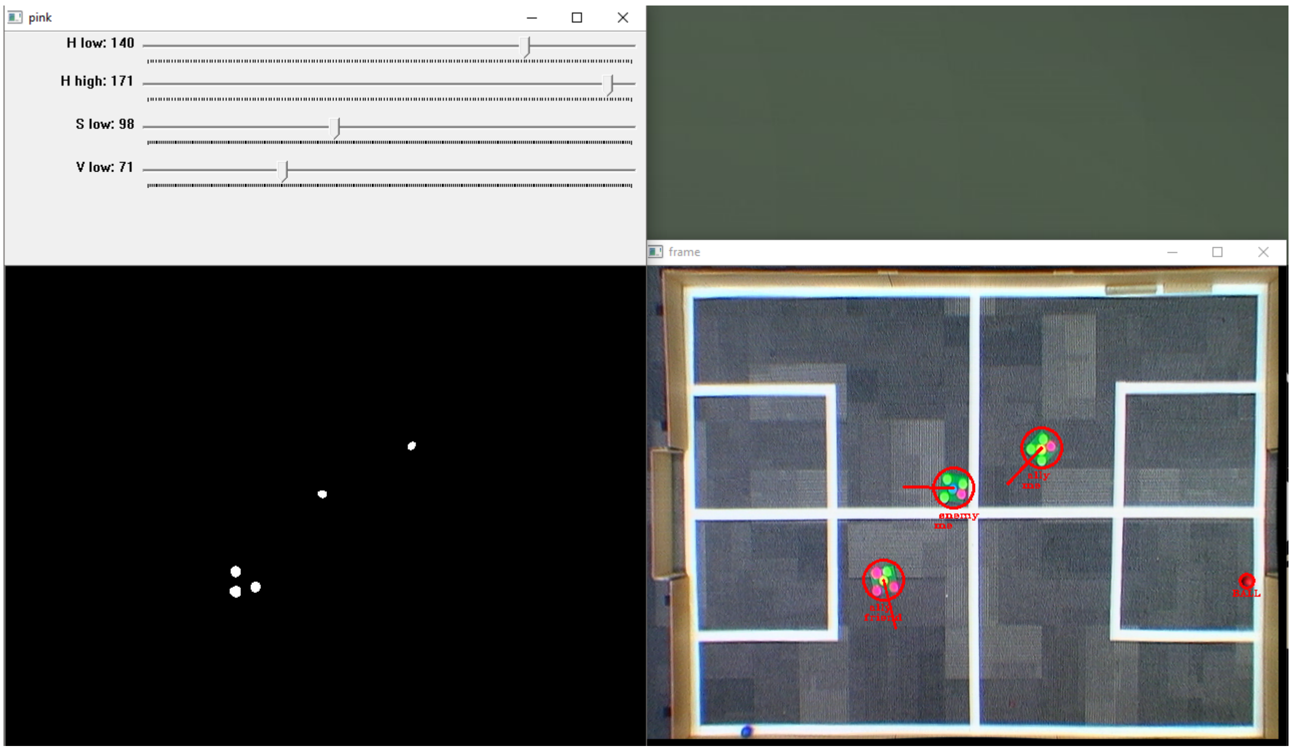
Each time the host machine is changed, the colour settings must be calibrated. This can be done by following these steps:

1. Run `$ xawtv` and manually adjust the brightness and contrast so that the colours are clearly visible (use Figure 2 as guidance).
2. Run `$ python vision/color_calibration.py -p [pitch number, either 0 or 1] -t [1 if live camera feed is available and 0 otherwise]`. If there is no blue ball on the pitch for calibration, just skip it, its colour values can be added later, but for the rest of the colors, top plates must be present on the pitch.
3. Type each colour's character (`b` for blue, `c` for light_blue, `p` for pink, `y` for yellow and `g` for green) and click on a couple of instances of this colour on the vision feed. If you make a mistake, type that colour's character again to restart for that colour. Once finished, press `Esc` to save the initial calibration.
4. After pressing `Esc` to proceed, a video feed and separate window with slide bars will appear. Change the `S_low` and `V_low` bars until all objects of that particular colour are visible in the mask. It is not recommended to change `H_low` and/or `H_high` values, though possible if needed. Once all objects in the mask are clearly visible (see Figure 2), pres `Esc` to proceed to another colour until all colours have been calibrated, at which point the camera feed will disappear and a calibration confirmation will be written to the terminal. The calibration settings are saved into `vision/config/colors.json`.

Note: color thresholds after calibration can be manually changed during the game once running `vison/sender.py` file, using the same method as the initial colour calibration.

4 Running the System

1. To begin, place the robots and the ball on the pitch. In order to configure the colour settings, you'll need to have at least one circle of every colour visible.
2. Ensure all planning configuration settings are correctly configured for this game (see **??**)
3. Run `$ python vision/color_calibration.py` to calibrate the vision (see § 3.2)



H low: 140
H high: 171
S low: 98
V low: 71

Figure 2: Pink colour mask and object classification in the vision feed

4. Plug in the RF stick and test it using the terminal command `$ ls /dev/ACM0`. This should return `ACM0`. If it returns nothing, or gives an error, there's an issue with the stick. See § 5.1 for troubleshooting.

5. Plug in the robot's batteries and reset the Arduino. The robot should spin on the spot for roughly two seconds before sitting still. If not, there's a problem with the batteries, motors or the arduino code: see § 5.1 for troubleshooting.

6. Test that everything is working correctly by running `$ python comms/CommsAPITest.py`. If all is well, this will make the robot move around slightly.

7. Reset the Arduino once more

8. In a separate terminal, run `$ python vision/sender.py -t [team colour] -m [robot colour] -b [ball colour] -p [pitch number] -test [optional - needed if no camera feed is available]` . This observes the pitch and sends information to the planner, plus brings up the annotated live camera feed, showing its belief about the positions of the robots and ball and each robot's orientation.

You are now ready to play. Run `$ cd planning && python robotAI.py` to begin playing

5 Troubleshooting

5.1 Issues

5.1.1 The robot will not move

Try changing the batteries. These are located at the top of the robot. To remove them, you disconnect them from the Arduino board, then remove the battery pack through the slot between the robot and the top plate holder. Swap out all of the batteries for fully charged ones and replace the battery pack, then connect the pack to the Arduino.

If the batteries are charged, simply try disconnecting and connecting them again. Sometimes, programming the arduino whilst the batteries are plugged in kills the radio before a hard reset is performed.

Otherwise, try unplugging and plugging back the RF stick. The RF stick tends to stop receiving or transmitting data if used for over 4 hours.

5.1.2 The robot moves sluggishly

Replace the batteries. See § 5.1.1 for instructions on how to do this.

5.1.3 The robot moves when it should be spinning on the spot

Check that the batteries are fully charged, as low charge can cause the motors to fail to turn. See § 5.1.1 for instructions on how to do this.

If this fails to fix it, one of the motors is not working properly. Find out which by lifting the robot and inspecting the wheels. Ensure the connections on this motor are all securely in the right ports.

5.1.4 Robot orientation in the vision feed is wrong

Some of the color calibration values are wrong. Find the color which is detected poorly by pressing the corresponding colour character on the keyboard, then re-calibrate it as described in § 3.2.

5.1.5 Vision system misclassifies objects

If `vision/sender.py` is running and objects are misidentified, then ensure the correct command line arguments (team and robot colours) were given. If so, `yellow` or `bright_blue` are not being detected correctly, so re-calibrate these as described in § 3.2.

5.2 Known Bugs

5.2.1 Vision System

- If a robot or ball is removed from the pitch, its position is still shown on the pitch. This is by design, as if the vision system loses an object, it predicts where it expects it to be based on its previous orientation and speed so as to avoid losing objects temporarily. This should not cause any problem to the planning system, and the object should be identified correctly once the object is placed back on to the pitch.

5.2.2 Hardware System

- The robot will continue to run for a few seconds once the strategy module has completed. This is due to the arduino-side command buffer not being empty upon termination. Simply reset the arduino or remove the batteries to stop the robot.