User Guide

SDP Group 12

March 7, 2016

1 Introduction

Once ready to use, place a charged batteries in the back holder, and connect it to the arduino power cable.

Plug the RF stick in the computer, ensure top plates of all types, as well as the ball, are on the pitch, and execute '`./main -p <PLAN> -1 <PATH> -c <COLOR>`', there `<PLAN>` is the plan to run (see section below), `<PATH>` is the device path of the RF stick (e.g. '`/dev/ttyACM0`'), and `<COLOR>` is the team color, which must be one of '`blue`', '`b`', '`yellow`', or '`y`' respectively. Further options notably include the '`-l`' option, which sets the logging level. E.g. '`-l info`' enables info messages in the logger.

While the control program is running, the overall strategy and logging can be modified. Entering '`debug`', '`info`', '`warn`', or '`error`', and pressing enter, sets the logging level appropriately. Likewise, entering a plan name and pressing enter switches the running plan to what was entered. Entering '`stop`' unsets the active plan and leaves the robot idle.

1.1 Plans

A small number of plans are available to run:

- '`move-grab`', to move to the ball and grab it.
- '`m1`', to do milestone 3, task 1.
- '`m2`', to do milestone 3, task 2.
- '`m31`', to do milestone 3, task 3.1.
- '`m32`', to do milestone 3, task 3.2.

2 Vision

2.1 Requirements

You'll need the following python packages to successfully run the vision:

`Polygon2` Polygon is a python package that handles polygonal shapes in 2D.

`argparse` Python command-line parsing library

`pyserial` Python Serial Port Extension

`numpy` Array processing for numbers, strings, records, and objects.

`openCV` OpenCV-Python is the Python API of OpenCV. It combines the best qualities of OpenCV C++ API and Python language.

To install them run these commands in the terminal:

```
# pip install --user Polygon2==2.0.6
# pip install --user argparse==1.3.0
# pip install --user pyserial==2.7
# pip install --user numpy
```

You can also learn how to install openCV from the following link: `http://docs.opencv.org/2.4/doc/tutorials/introduction/linux_install/linux_install.html`

2.2 Usage

Before using the vision system, you can have a look at the vision feed by typing `xawtv` in the command prompt. This will launch only the vision feed, where you can experiment with the different settings.

In order to launch our vision system, you'll need to run the main vision file (python vision.py). At first, a window for the automatic colour calibration will pop out. You'll need to follow the instructions as printed in the terminal. The calibration goes through all the colours that are used for the vision (textttred, textttyellow, textttblue, textttgreen, textttpink) and requires multiple clicks for each one of them to get their thresholds. You need to press the textttq button after each calibrated colour. If you want to skip the calibration you can simply press the textttEsc key and the vision will use the previously saved calibrations.

After this, the vision will be launched. There will be a window named textttFilter output, where you can see the vision feed with objects drawn on it representing the robots and the ball (if they are found). Robots will be represented by an inner circle for the team colour (textttyellow or textttblue), an outer circle identifying which of the two team mates it is (textttpink or textttgreen) and an arrow giving the direction of the robot. The red ball is simply shown by drawing a red circle around it. You'll also notice two other windows containing several trackbars. These trackbars are for filters that you can add to the output feed. There are filters to show only specific colours, or to different effects to the frame.

When the vision is running, it will provide the coordinates of all found objects, their orientation and velocity relative to the previous taken frame. These objects are returned as a dictionary and are passed onto the planner. This is achieved by passing a method for updating the planner world model when initialising the vision module. This is then called on every vision update, and it ensures the planner has the latest data.

3 Planning

3.1 Running the planner

The planner can be run as part of the system from the command line. This is done using:

python main.py [-2PATH] OPTIONS TEAM-COLOUR -g GOAL-END

where TEAM-COLOUR refers to the colour of the controlled team's top plate and can be either -y or -b (yellow or blue) and OPTIONS can be –debug Set logging level to 'debug' –info Set logging level to 'info' –warn Set logging level to 'warn' –error Set logging level to 'error' and GOAL-END refers to the controlled team's goal end, either 'left' or 'right'

4 Hardware

4.1 Changing the batteries

The batteries can be change by detaching the battery packs from each other and snapping off the ribbon connectors, then replacing the batteries in each holder. They should be charged with the nitecore intellicharger. The battery holders should snap back on with the lego connectors and plug back into each other, and the ribbon connectors.

4.2 Turning the robot on

Flip the switch marked 'ON' to the 'ON' position.

5 Arduino Software

5.1 Sending commands

The commands should be sent through the planner with the RFComms.py module.

5.2 Command Set

The commands are as follows: kick(distance in cm) grab() release() turn(angle in deg) move(distance in mm)

5.3 Command Response

In response to a command the robot immediately drops what it is doing and runs that command.

5.4 Kicking

The robot releases the grabbers until they are at position 0 then kicks a time dependent on the distance required. It then closes the grabbers to position 13 (fully closed) for continued movement.

5.5 Grabbing

The robot closes the grabbers until they are fully closed or 800ms whichever is sooner, if the grabbers only close to position 10 the grab is considered a success and this is sent to the planner.

5.6 Release

The grabbers are released to position 0.

5.7 Turn

The robot accelerates up to a calibrated turning speed, then maintains that speed until it has just enough time to decelerate to the required position. The wheels are kept turning the same distance using the distance from the rotary encoders by powering down a motor if it has gone too far and powering it up if it has not gone far enough. The turn is considered finished when the averaged distance from the left and right wheels matches the calculated distance required to rotate the desired angle (by using radius of wheels:2.5cm, radius from origin:7.5cm).

5.8 Move

The robot accelerates up to the calibrated speed and then maintains that speed until it has just enough time to decelerate to the desired distance. It keeps the wheels going the same distance using the rotary encoders by reducing power to a wheel that has gone too far. The distance is determined by the radius of the wheels * pi * rotation.