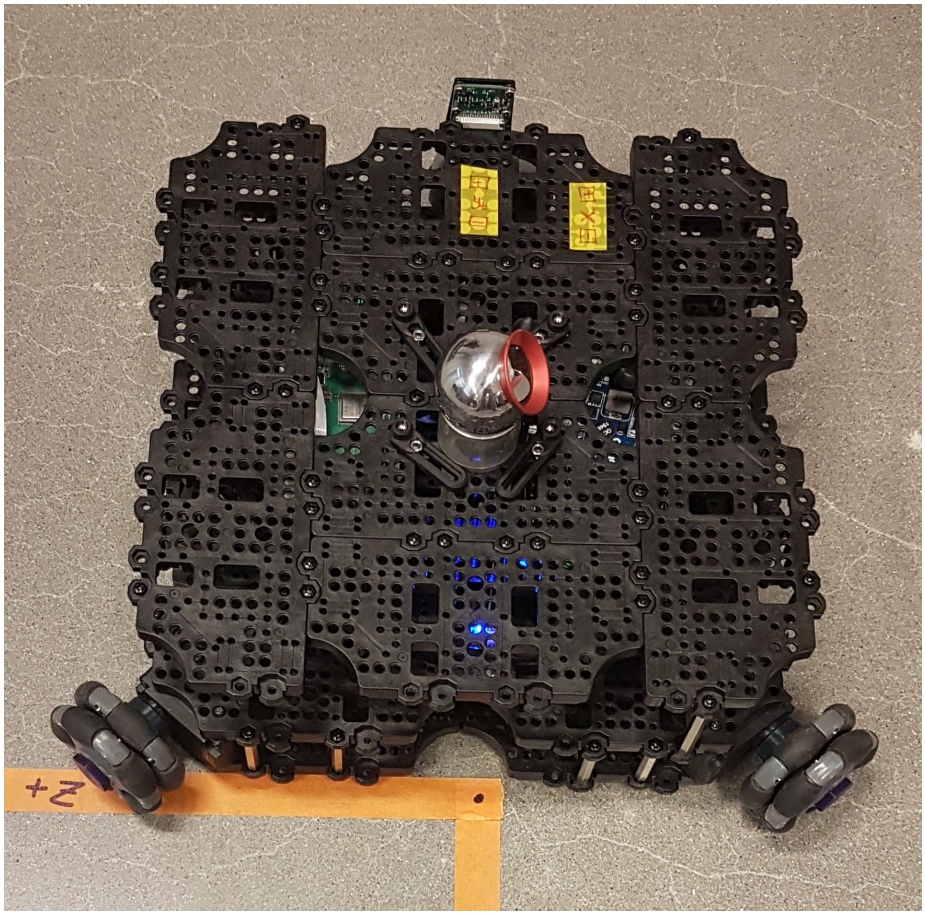
FRTF20 - bluelining maxIV

Haapamäki, Kim
ki1875ha-s

Berntsson, Jacob
ja8421be-s

Jeppsson, Staffan
st2887je-s

October 2020



Contents

1 Abstract

The main goal of the project was to deliver a robot capable of navigating with the help of a laser to a desired point with high precision without loosing the connection to the laser. To our aid we had a TurtleBot3 which we modified with corresponding software and hardware to be able to run with omniwheels, be able to connect and recieve messages from the Leica absolute tracker and use the information to move the robot. The robot was able to move towards the target point from different starting locations ending up within a margin of 4 cm. The robot were able to achieve even better results such as a margin of 2 cm without any difficulties.

2 Introduction

The need for tools that are precise and trustworthy is getting more and more apparent. For instance, when installing equipment used in experiments. When installing the equipment a margin of one micrometer can be the difference between a successful test and a failed one, and in some cases the margin might be even smaller. At MaxIV the need of such a tool is no less. Currently, the installment of equipment at MaxIV is carried out manually by humans. The goal of this procedure is to mark the floor with blue markings, with a very high precision, where the equipment should be installed. The positions where the different markings should be placed has already been defined and has been added to the coordinate cloud, which is a representation of MaxIV in a coordinate system. The procedure is as follows. A Leica Absolute Tracker, which is a portable coordinate measuring machine, is positioned in the room where the equipment should be installed. This laser is able to automatically track and follow a reflector as long as the Leica has found the reflector and the reflector does not move outside the 60 degree field of view or is moved in a to fast motion. When the connection between the Leica laser and the reflector is established the Leica laser gives feedback, consisting of x, y and z coordinates, of how far the reflector is from the position that should be marked on the floor. The human then have to carefully and in small motions move the reflector to the desired location. This is tiring and tedious work since you have to be on the ground. When the desired position has been found, i.e the reflector is within a satisfied margin of the position, the reflector is removed and a blue pen is used to mark the position on the floor. Since the positioning of the reflector demands such a high precision, this work is not suitable for humans. All it takes for the operator of the reflector to ruin the calibration is, for instance, a sneeze or trembling hands. Therefore, the need for a robot that is able to navigate and locate the positions and also mark the positions with high precision is needed. We have been assigned the task to create a robot that is able to navigate towards the desired positions and locate them, without loosing the connection with the laser, and place a manipulator that is going to mark out the position with a 3D printer. The robot should be able to place the manipulator within a margin of

a couple of centimeters or better.

3 Experimental

3.1 Hardware

In order to construct a robot that will be able to solve the problem stated above, the TurtleBot3 waffle kit was not enough. Some extra hardware were needed in order to make the robot solve the task, such as omniwheels for moving without rotation etc. Here we will list the ones that were used in the project.

1. TurtleBot3 equipped with:
    1.1. TurtleBot3 waffle kit
    1.2. Raspberry Pi 3 Model B+
    1.3. OpenCr 1.0
    1.4. 3x XL430-W250-T Dynamixel motors
    1.5. 3x Omniwheels
    1.6. Imu
    1.7. LI-PO 11.1V Battery
2. Ohers
    2.1. WiFi Router
    2.2. Leica absolute tracker with corresponding reflector
    2.3. Remote PC

3.2 Software

3.2.1 ROS

In order to create robust software for our TurtleBot the Robot Operating System (ROS) was used. It is a flexible framework for writing software for your robot. The ROS distribution that was used were Kinetic Kame, since it is primarily targeted at the Ubuntu 16.04 release.

3.2.2 OS/VirtualBox

Since it was stated that ROS currently only runs on Unix-based platforms, such as Ubuntu and Mac OS, the choice of operating system were Ubuntu 16.04. Since a desktop PC with windows were at our disposal we could set up a virtualbox on the PC. The program used to set up the virtualbox were Oracle VM VirtualBox.

3.2.3 Dynamixel

When installing new Dynamixel motors they need to be calibrated, for this purpose we used the Dynamixel Wizard. A more detailed view over this process can be found in Appendix.

3.2.4 OpenCR/Arduino

The OpenCR is a control module which has been developed for ROS embedded systems which contains microcontrollers that can be programmed. Arduino was used in order to upload firmware to the microcontroller, which from the get go contained firmware for the TurtleBot3 core. In order to make the robot able to use the omniwheels the TurtleBot3 core firmware had to be modified. The Arduino IDE contained firmware for the TurtleBot3 omni wheels, unfortunately, this firmware was only for testing the omni wheels with the Robotics RC100 controller. So a modified TurtleBot3 core firmware was created with an addition of logic able to control the motors when handling omni wheels. A description of how to upload TurtleBot3 core firmware for a turtlebot3 waffle with Arduino IDE can be found in Appendix

3.3 MaxIV and Bluelining

In MaxIV there is a database with thousands of 3d points that describe the building. These points are both physical points in MaxIV that are used to calibrate the Leica absolute tracker but also points where new equipment are to be installed.

The points that are defined for new equipment are used in the task of bluelining, this is the action of marking the 3d points in the physical enviroment as blue dots on the floor of MaxIV. This is currently done by hand with the Leica absolute tracker and corresponding reflector. With the suggested solution a robot will instead do this task, by having a larger robot act as taxi for a manipulator. The manipulator will then do the final adjustments and mark the floor with the dot for later installation of equipment.

4 Result

Our program achived the results of being able to connect and receive messages from the Leica absolute tracker, use this information to make decisions to move towards the goal target and send these actions to the motors of the robot to behave accordingly.

Our robot is able to keep its orientation using a built in inertial measurement unit(IMU) to keep the reflector somewhat pointing to the Leica absolute tracker. This is necessary due to the high drifting caused by the omniwheels that lack friction on some floors and the not optimal structure of the robot.

The robot was also capable of calculating the correct transforms from local coordinate system to the global to be more flexible in the start position of the robot.

Our result ended with the robot being able to move towards the target point from different start locations, ending up without any trouble inside of a radius of 4cm that is specified from the manipulater to be sufficient. Our testing also showed that the robot is capable of going closer with 2cm without any big difficulties.

5 Discussion

To achieve our results we had to narrow the objective to be reasonable with the time we had working on this project but would be necessary for the project to be stable and optimal.

Currently the robot is able to navigate around keeping its orientation, negating the drifting caused by the omniwheels and structure of the robot. However this helps it while inside of the 60 degrees of freedom from the but going outside is not possible with the current program. However if the information is gained from the manipulator with orientation of the robot with regards to the Leica absolute tracker it is possible to maintain a orientation with the reflector towards the Leica absolute tracker. Since we only have one point of the robot and the Leica absolute tracker it is impossible to calculate this orientation with accuracy due to drifting. We would need the robot to move from one point to another with knowledge it actually moved in a perfect motion to know that the orientation is correct. This is however something the manipulator can achieve.

The built in inertial measurement unit(IMU) is affected by magnetic fields close by the robot and therefore a source of unexpected errors. In MaxIV there are many equipment's that disturb the global magnetic field, this causing the robot to be confused in some situations. However this was not a problem when testing the robot in MaxIV but might be in some situations. If the orientation is instead given from the manipulator using the Leica absolute tracker this would not be a problem. Also the IMU is set to 1 in orientation when the robot starts fresh, a value going towards -1 when rotated 360 degrees in both directions. This is also a problem due to the robot needing to know what rotation corresponds to negative positive resulting orientation. This caused us to define a manual when running the program to state the robot has to be started in a few degrees from intended orientation, and be moved after startup to the intended orientation for the IMU to be correctly used.

The robot has its own local coordinate system and the Leica absolute tracker is connected to the coordinate system defined in MaxIV. Our program is only capable of moving the robot in 4 directions for now, forward, backward, right and left. This casues the issue of needing the robot to be in correct orientation with regards to the MaxIV coordinate system. Our program is able to calculate the transform if a positive movement in local X to a positive/negative movement in MaxIV coordinate system. Also the same for local Y to the MaxIV Z. So

a requirement we put down to make our work simpler and faster to achieve the results we got, the starting requirement of placing the robot in the correct orientation of MaxIV X, Z coordinate axes was created. However with the omniwheels we are able to move the robot in any possible direction, so with more time it would be possible to create a initial function that calculates correct movements in any starting position.

In our project the coordinate system of Max IV was made to have X, Z forward/left and Y up, the local coordinate system of the robot is X, Y forward/left and Z up. This could be changed easily in the Leica absolute tracker. But needs to be defined in the code if changed.

6 Appendix

6.1 Code structure

This program is written in standard rospy style and it is highly encouraged to read and understand rospy before trying to understand our code. Below the main points of the program is described.

The main code is written in 3 separate files, main.py, action_service.py and laser_service.py. These files can be found in the git repository "here" under the folder src. This code structure and the flow of the program is showed in the figure 1

6.1.1 Main

Main file contains logic to connect the different parts of the program and send the resulting actions to the robot. It is composed of one initial function init() and one while loop that runs for the period the robot does the bluelining task.

The initial function init() tries to calculate the correlation of the x, y axis of the local space and the global. It also calculates the positive rotation given by the equipped inertial measurement unit(IMU) on the robot. With moving small steps in x, y while having information of the global space from laser service and rotating around the z axis the signs for the environment are calculated.

The main loop does 4 steps each iteration, first it will ask the laser service where the robot is supposed to go. This information is sent forward to the action service that responds with a direction or stop signal. Given this information the robot checks if the drift has altered the rotation to keep it directed towards the maxIV laser. After which the robot will be given the instructions to send to the 3 different motors to manipulate the position.

6.1.2 launch

In our program we run the code by launching a launch file. This is a way to start multiple nodes in one command making the setup easier. The launch file is located in the launch folder and specifies which nodes are to be run and what names they are given.

6.1.3 srv messages

Our code uses modified messages to communicate between the nodes, these are specified in the srv folder and contain the message structure of request/responses from both the action service and the laser service.

6.1.4 Action

Action service takes the distances the robot needs to move in x, y axis to reach its destination. Given these values it responds with a action.

6.1.5 Laser

Laser service has a UDP_connect class that creates a socket which takes messages sent to the robot on a specific port. When the laser service is called it will read the latest messages of position and distances to go and parse them to be returned to the main program.

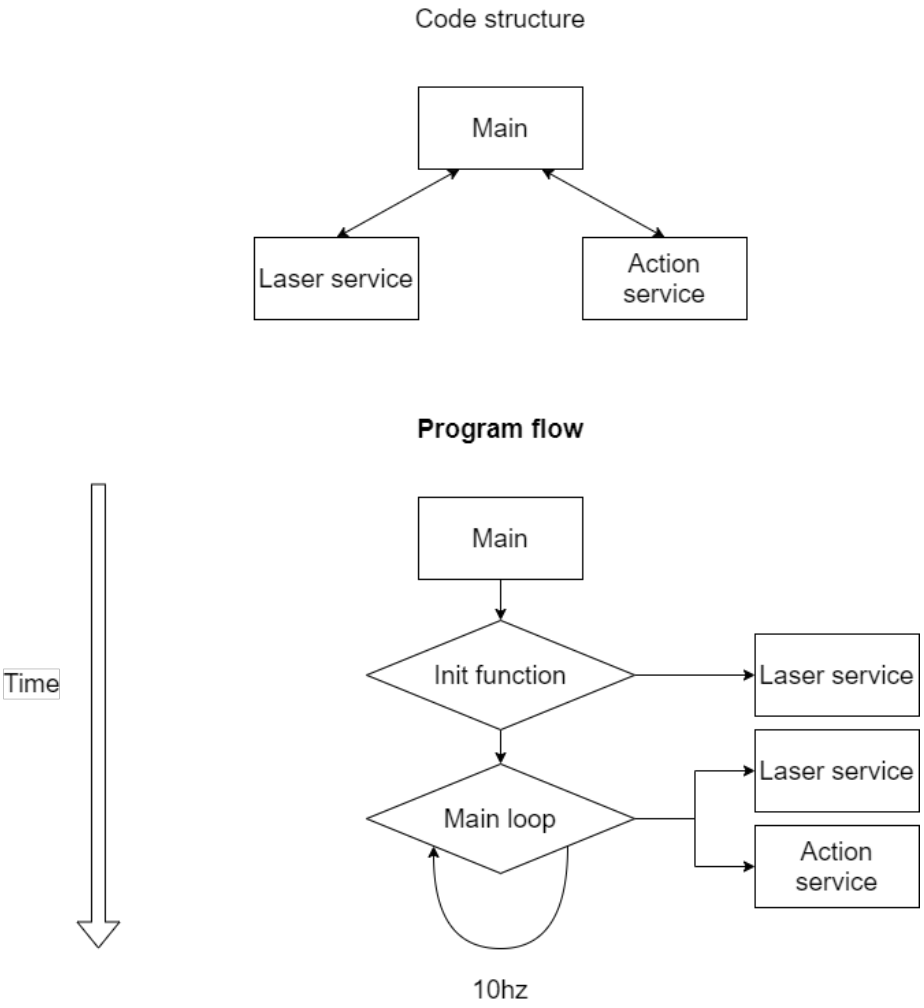


Figure 1: UML and program flow

6.2 Manual: running the robot

This manual specifies the settings to be used when having a remote PC running Ubuntu 16.04 and the raspberry PI in the currently used robot.

6.2.1 Arduino: omnisettings

By following the instructions Upload firmware with Arduino IDE we are able to upload the code written by us to give the robot capabilities of running 3 motors in omniwheel settings. The files we want to upload to the Arduino is found in our repository "here" under the folder arduino/turtlebot3_core. So by connecting to the Arduino on the robot and opening the .ino file in the project

we are able to upload our settings to the Arduino. For help to download the repository one can follow git basics manual "Getting a Git Repository"

6.2.2 WIFI

Since the robot is supposed to talk to a remote PC and the Leica absolute tracker there needs to be a easy connection between the devices. This is solved by a router, in our equipment we have a router that has defined static IP to the raspberry PI and the remote PC used by us in our lab. However if in a new environment it is needed to check a few settings. Connecting the raspberry PI and remote PC to the same wifi the IP needs to be checked and made sure it is defined in .bashrc file in the home folder (∼/.bashrc) of both raspberry PI and the remote PC. The settings needs to be defined as in figure 2. When these settings are defined they need to be loaded by "source ∼/.bashrc" or a new terminal needs to be opened.

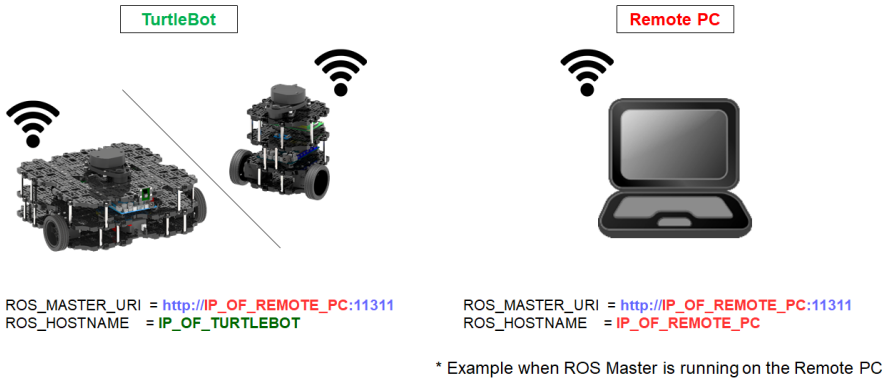


Figure 2: IP settings of TurtleBot and remote PC

6.2.3 Remote PC

Running the robot it needs a PC that does the calculations. Requirements of this PC is to have python2 and ROS Kinetic. Documentation to do this installation you can follow the following documentation from ROS, Ubuntu: Install ROS Kinetic. Then by running "roscore" in the terminal of the remotePC the connection to the remote PC is setup and will be used as the calculation service by the robot.

6.2.4 TurtleBot

In the TurtleBot there is a raspberry that will run the code of the program. It will be connected to the remote PC and send the correct instructions to the arduino that sends the commands to the motors that navigate the robot.

Further it is needed that we have the code of our program in the TurtleBot, this step will not be described as it is already in the robot in ∼/catkin_ws/src. If this needs to be setup from scratch on a new raspberry PI this ROS documentation can be used for referenced "ROS create a workspace". The code needs to be placed in the catkin_ws/src folder and then compiled with the given commands.

When the code and all other setup is done the robot is connected by running "roslaunch turtlebot3_bringup turtlebot3_robot.launch" and then in another terminal on the raspberry PI our code can be executed by running "roslaunch FRTF20bluelinigmaxIV launch.launch".

When the program is running it will first initialize and calculate the local and global differances and its orientation, after this it will wait for the rostopic /start to be set to true. This can be achieved by writing the following command in another terminal "rostopic pub /start std_msgs/Bool "data: true"" and by setting it to false it will stop the robot.

6.2.5 Demo of robot movement

In our code we have another launch file called demo.launch, if everything is setup with roscore on remote PC and turtlebot3 bringup on the robot we can demostrate movements of the omniwheels by running following command in a terminal on the raspberry PI by executing "roslaunch FRTF20bluelinigmaxIV demo.launch".

This code starts a topic called /demo, which we can call by "rostopic pub /demo". By using the [TAB][TAB] in terminal it will try to auto complete the command. This way we can get the Here we can define what we want the robot to to for 2 seconds before going to a stop again.

6.3 Configuration of Dynamixel Servo Motors

In order to install new motors for your TurtleBot they need to be configured. This section will go over how we did it for our project.

6.3.1 Components

In order to change the Dynamixel settings you need to connect the motor to your PC. This is done via a U2D2 module. The U2D2 module does not provide power to the Dynamixel motor, therefore you need an external power supply, the recommended voltage for each Dynamixel motor may vary by model. In our case the XL430-W250-T Dynamixel motors required a 12V power supply. This setup can be seen in Figure 3.

Figure 3: PC to Dynamixel setup with a U2D2 module

6.3.2 Settings

There are multiple tools that can handle the configuration and testing of Dynamixel motors. The program used during this project is the Dynamixel Wizard. Once the wizard is open you can make contact with the Dynamixel motor by scanning. The first time you do this you might be prompted to chose scanning options, if you are not sure what options your motor has make sure to scan everything. Once the Dynamixel motor has been found you are able to alter it's configuration and test it. The settings altered in this project were the ID, Baud Rate and Operating mode, but depending on your motors some settings may be different. Common for all motors is that every motor needs a unique ID. Furthermore each motor needs to have the same Baud Rate and Operating mode. The settings used during this project can be seen in Figure 4.

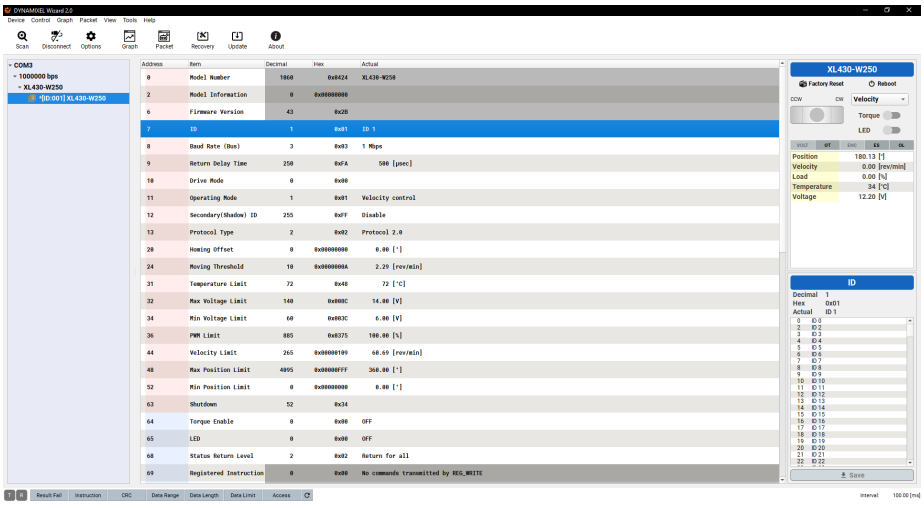


Figure 4: Dynamixel Wizard Settings

6.4 Upload firmware with Arduino IDE

In order to upload firmware with Arduino IDE the OpenCR USB port must be able to upload Arduino IDE programs, port to the Arduino IDE. The Arduino

IDE need to have the OpenCR board package installed and be able to connect to the OpenCR via the USB ports.

6.5 Upload

If the requirements stated above are fullfilled you can open the TurtleBot3 core firmware by selecting file → Examples → turtlebot3 → turtlebot3_waffle → turtlebot3_core

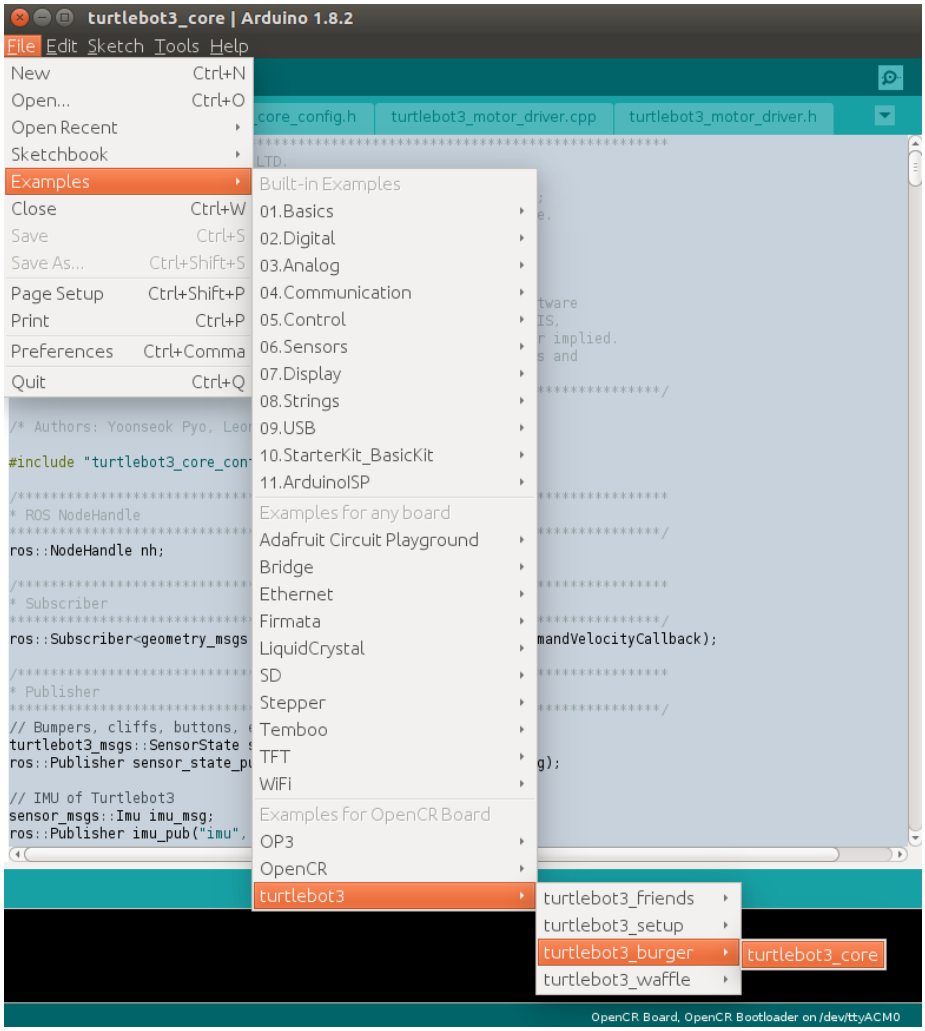


Figure 5: Arduino firmware

Click upload button to upload the firmware to OpenCR.

When firmware upload is completed, a jump_to_fw text string will be printed on the screen.

Now the OpenCR is updated with the new firmware.

(a) Arduino upload process. (b) Arduino upload complete.