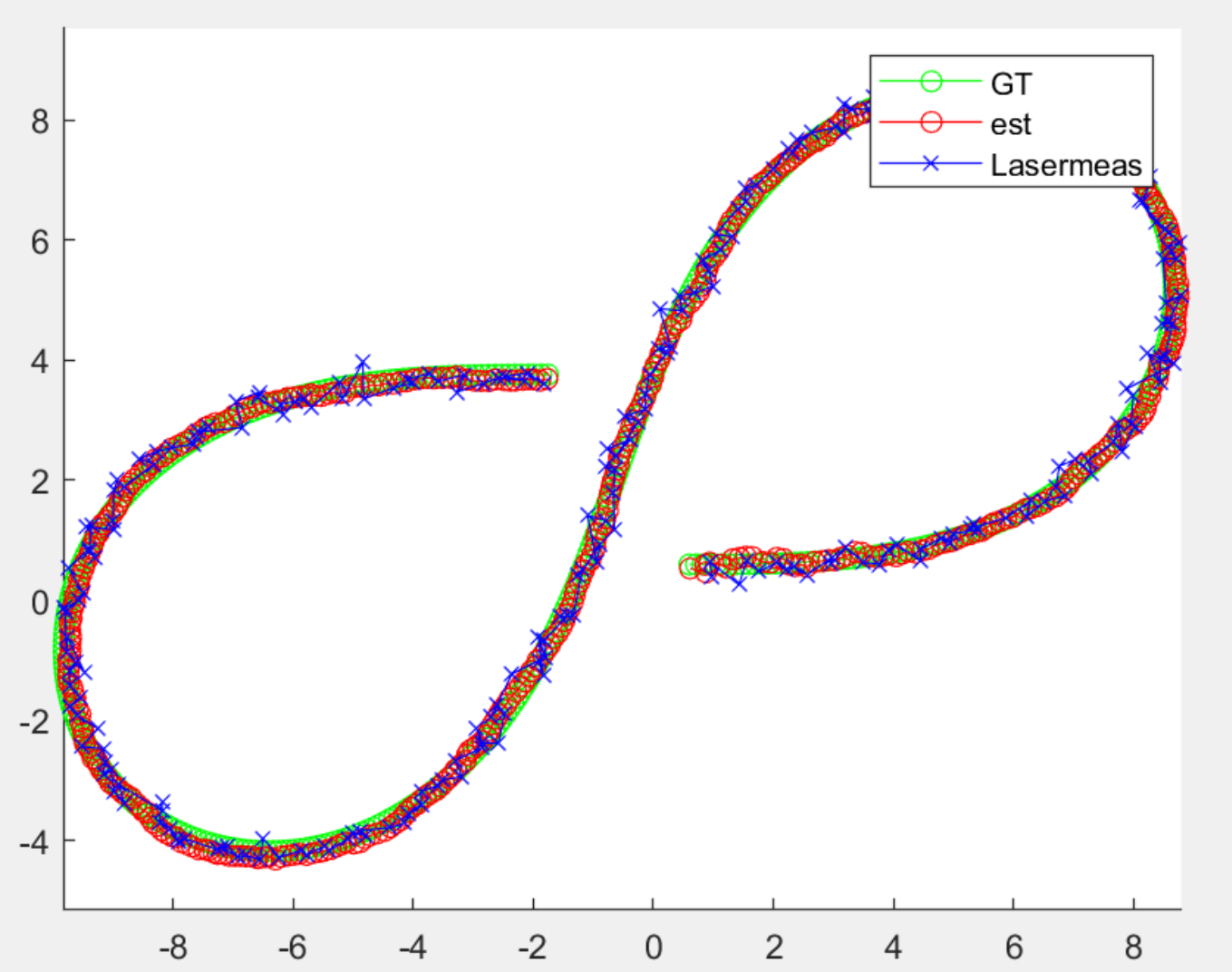
Capstone Project for C++ Specialization

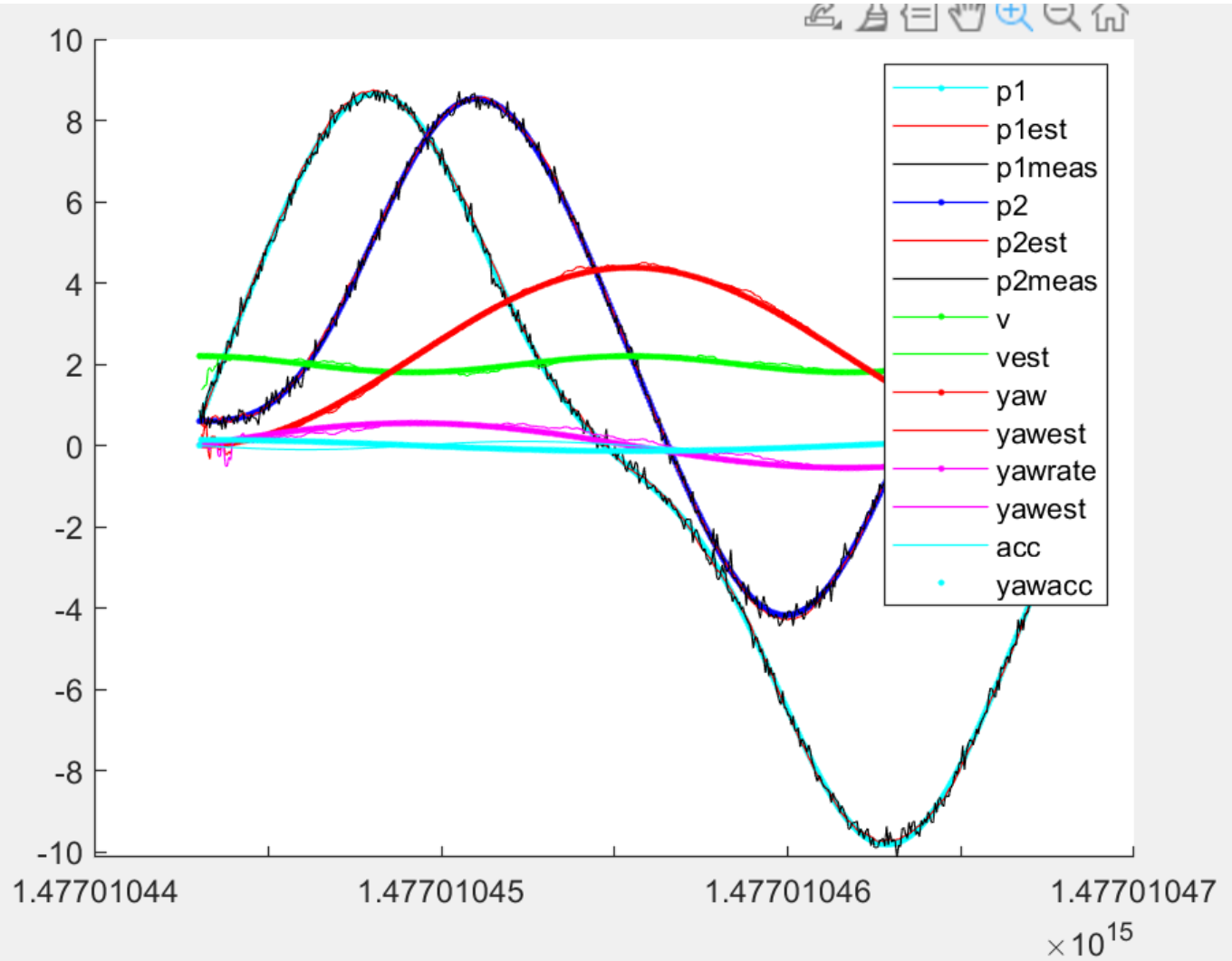
Overview

This project make use of an Extended Kalman Filter (EKF) to estimate the state of a moving object of interest with noisy LIDAR and RADAR measurements to detect a bicycle that travels around a vehicle. The project uses an Extended Kalman Filter, fusion of LIDAR measurements, RADAR measurements to track the bicycle's position and velocity. The final project can be seen here.

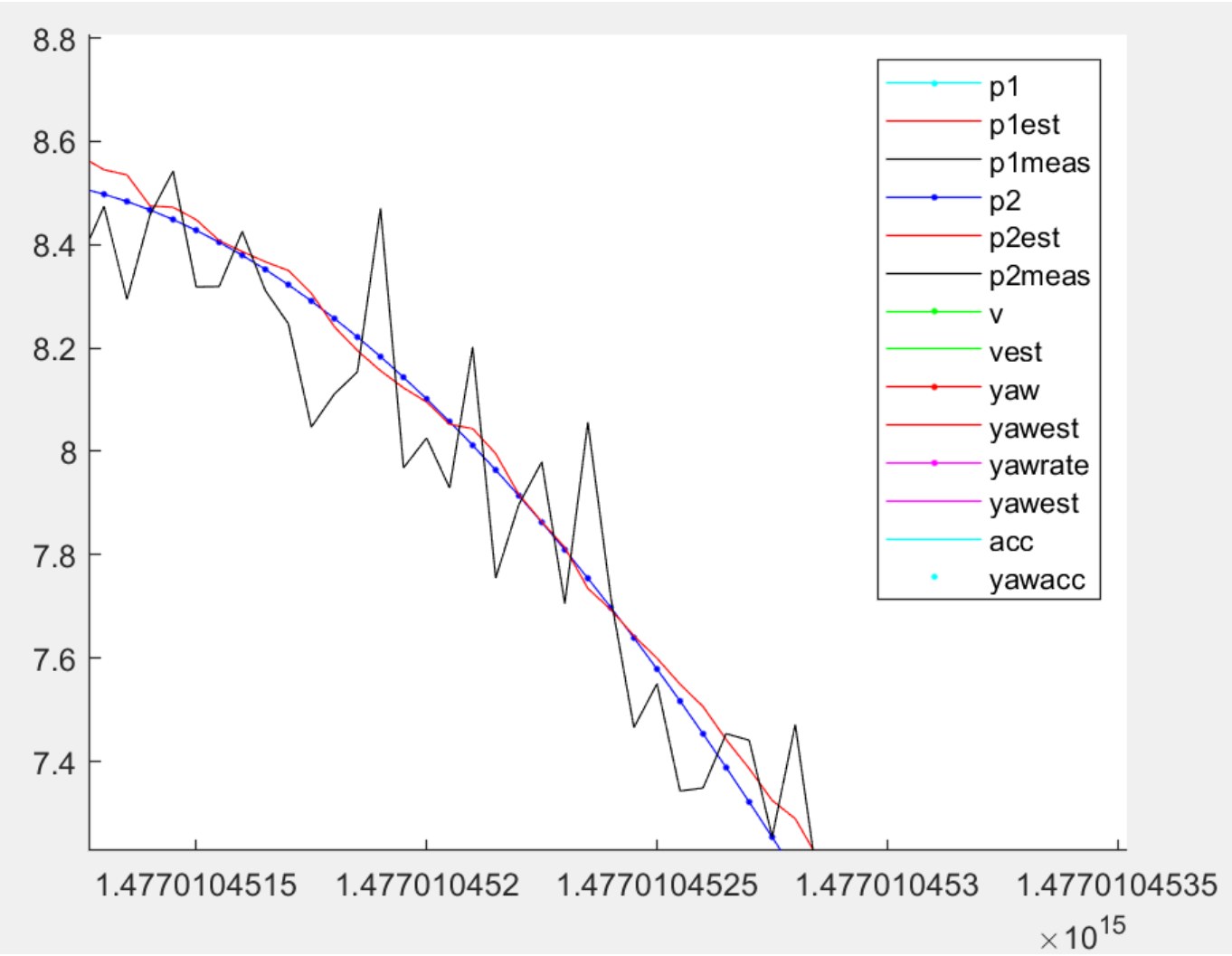
The figure shows the complete trajectory of vehicle.



The figure shows how the predicted values are close to ground truth values.



The figure shows details of image above.



Instructions to install

The first step is to download the simulator that will simulate the lidar and radar sensors sending measurements that are processed by the EKF algorithm. The simulator can be download from here.

This repository includes two files that can be used to set up and install uWebSocketIO for either Linux or Mac systems. For windows you can use either Docker, VMware, or even Windows 10 Bash on Ubuntu to install uWebSocketIO.

uWebSocketIO Starter Guide

This project involve using an open source package called uWebSocketIO. This package facilitates the connection between the simulator and the C++ code. The package does this by setting up a web socket server connection from the C++ program to the simulator, which acts as the host. In the project repository there are two scripts for installing `uWebSocketIO` - one for Linux and the other for macOS.

Note: Only uWebSocketIO branch e94b6e1, which the scripts reference, is compatible with the package installation.

Linux Installation:

From the project repository directory run the script: `install-ubuntu.sh`

Mac Installation:

From the project repository directory run the script: `install-mac.sh`

Some users report needing to use `cmakepatch.txt` which is automatically referenced and is also located in the project repository directory.

Windows Installation

Although it is possible to install uWebSocketIO to native Windows, the process is quite involved. Instead, you can use one of several Linux-like environments on Windows to install and run the package.

Bash on Windows

One of the newest features to Windows 10 users is an Ubuntu Bash environment that works great and is easy to setup and use. Here is a nice step by step guide for setting up the utility.

I recommend using the version of Ubuntu Bash 16.04 or 18.0, which is able to run the `install-ubuntu.sh` script without complications. The link here can help you check which version of Ubuntu Bash you are running, and also help you upgrade if you need to.

Docker

If you don't want to use Bash on Windows, or you don't have Windows 10, then you can use a virtual machine to run a Docker image that already contains all the project dependencies.

First install Docker Toolbox for Windows.

Next, launch the Docker Quickstart Terminal. The default Linux virtual environment should load up. You can test that Docker is setup correctly by running `docker version` and `docker ps`.

You can enter a Docker image that has all the project dependencies by running:

```
docker run -it -p 4567:4567 -v 'pwd':/workspace/controls_kit:latest
```

Once inside Docker you can clone over the GitHub project repositories and run the project from there.

Port forwarding is required when running code on VM and simulator on host

For security reasons, the VM does not automatically open port forwarding, so you need to manually enable port 4567. This is needed for the C++ program to successfully connect to the host simulator.

Port Forwarding Instructions:

1. First open up Oracle VM VirtualBox
2. Click on the default session and select settings.
3. Click on Network, and then Advanced.
4. Click on Port Forwarding
5. Click on the green plus, adds new port forwarding rule.
6. Add a rule that assigns 4567 as both the host port and guest Port.

Other Important Dependencies

- cmake >= 3.5
  - All OSes: click here for installation instructions
- make >= 4.1 (Linux, Mac), 3.81 (Windows)
  - Linux: make is installed by default on most Linux distros
  - Mac: install Xcode command line tools to get make
  - Windows: Click here for installation instructions
- gcc/g++ >= 5.4
  - Linux: gcc / g++ is installed by default on most Linux distros
  - Mac: same deal as make - install Xcode command line tools
  - Windows: recommend using MinGW

Basic Build Instructions

1. Clone this repo.
2. Make a build directory: `mkdir build && cd build`
3. Compile: `cmake .. && make`
   - On windows, you may need to run: `cmake .. -G "Unix Makefiles" && make`
4. Run it: `./ExtendedKF`

This repository has a `Makefile` and the instructions above you can simply replace by the following commands:

1. make clean
2. make build
3. cd build
4. ./ExtendedKF

Source Files

This project has the following source files located in the `src` directory: main.cpp, FusionEKF.cpp, FusionEKF.h, kalman_filter.cpp, kalman_filter.h, tools.cpp, tools.h, measurement_package.h, json.hpp.

Protocol Communication

Here is the main protocol that main.cpp uses for uWebSocketIO in communicating with the simulator.

INPUT: values provided by the simulator to the c++ program

["sensor_measurement"] => the measurement that the simulator observed (either lidar or radar)

OUTPUT: values provided by the c++ program to the simulator

["estimate_x"] <= kalman filter estimated position x

["estimate_y"] <= kalman filter estimated position y

["rmse_x"]

["rmse_y"]

["rmse_vx"]

["rmse_vy"]