Lab Guide
State Estimation

Background

The objective of the state estimation lab guide is for students to develop an understanding of different state estimation techniques and their design considerations. Prior to starting this lab, please read through the following concept documents:

- Concept Review – Kalman Filters
- Concept Review – Kinematic Bicycle Model

Considerations for this lab guide:

State estimation is highly dependent on how the frame of reference of the moving body is defined. Concept Review – Kinematic Bicycle Model uses the center of the body as the frame of reference to derive the motion model for a vehicle. Quanser's implementation of the state estimation pipelines use the rear wheel as the origin of the body frame for the QCar. Students are encouraged to try out both methods to understand the difference between shifting the body frame origin of the QCar.

The artificial GPS information which the QCar will have access to broadcasts position and orientation information for the center of the back axel of the QCar. Students who would like to develop a state estimator which uses the center of the QCar as reference will need to apply a geometric transformation to this information.

This lab will ask you to complete 3 pipelines as shown below:

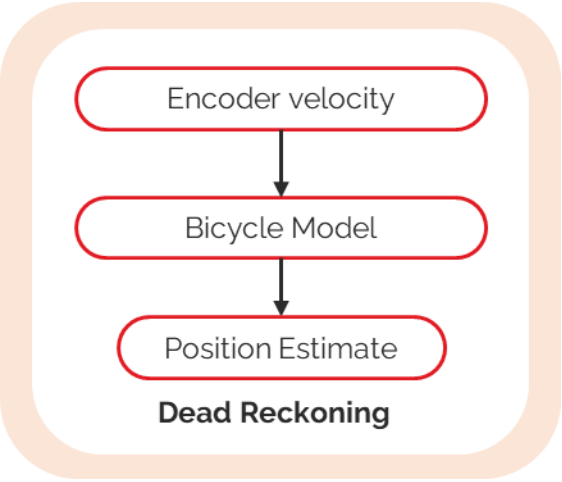

Figure 1: Dead Reckoning Based Kalman Filter

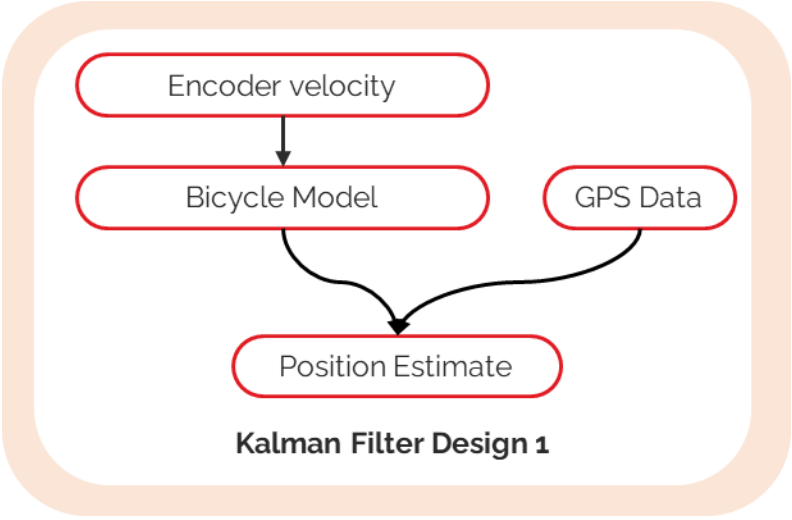

Figure 2: Kalman Filter

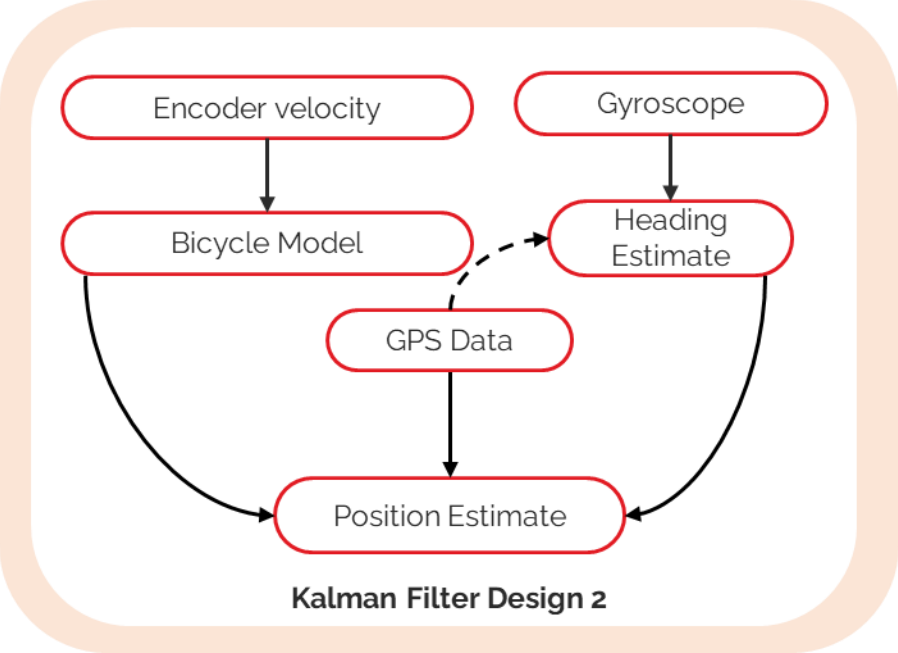

Figure 3: Coupled Kalman Filter Design

Skills Activity - Setup

The QCar will sample the tachometer data at the speed specified by the control. Variable **tf** will dictate the length of the state estimation example. Variable **controllerUpdateRate** will be used to define the sample rate for the state estimation controller used in this lab. It's only required to define these variables once.

```
#================ Experiment Configuration ================
# ===== Timing Parameters
# - tf: experiment duration in seconds.
# - controllerUpdateRate: control update rate in Hz. Shouldn't exceed 500

tf = 10
controllerUpdateRate = 100
```

Skills activities will focus on the algorithms used for state estimation.

Skills Activity 1 – Kalman Filter Dead Reckoning

The objective of this section is to implement the bicycle model for the QCar as the starting step for a discretized implementation of Kalman Filter state estimator. The motion model prediction will run as fast as the QCars' tachometer data can be read. The tachometer read rate is the same as the **controllerUpdateRate** specified in the setup section.

In this lab, you will carry out the following steps,

1. Implement the kinematic motion model for the QCar

2. Calculate the Jacobian for the motion model
3. Derivate the predict portion of a Kalman Filter state estimator

Step 1 – Implementing the Kinematic Bicycle Model

Complete **SECTION A** of the **QCarEKF()** class. The goal of this method is to take the previous state $\boldsymbol{X} = [x, y, theta]$, system input $\boldsymbol{u} = [v, \delta\ ]$ and time step dt to propagate the states of the QCar over time.

```
    # ==============  SECTION A -  Motion Model ====================
    def f(self, X, u, dt):
        # Kinematic Bicycle Model:
        # - X = [x, y, theta]
        # - u[0] = v (speed in [m/s])
        # - u[1] = delta (steering Angle in [rad])
        # - dt: change in time since last update

        return X
```

Step 2 – Implementing the Jacobian for the Kinematic Bicycle Model

Extended Kalman Filters make use of the motion model Jacobian to propagate the uncertainty of a system over time. Based on the motion model implemented in **SECTION A** calculate the corresponding Jacobian matrix. The inputs to the Jacobian class **Jf** are $\boldsymbol{X} = [x, y, theta]$, system input $\boldsymbol{u} = [v, \delta\ ]$ and time step dt.

```
# ==============  SECTION B -  Motion Model Jacobian ====================
def Jf(self, X, u, dt):
      # Jacobian for the kinematic bicycle model (see self.f)

      return np.eye(3)
```

Step 3 – Combining the motion model and Jacobian to predict the state of the QCar

Using the equations for state prediction for an Extended Kalman Filter Combine the outputs for **SECTION A** and **SECTION B** to generate a dead reckoning based state prediction. The exact implementation being performed is a discretized Extended Kalman Filter.

```
    # ==============  SECTION C -  Motion Model Prediction ====================
    def prediction(self, dt, u):
        '''
        return
```

Physical setup -

Place the QCar on the floor ensuring there are no obstacles within a 1m radius of the QCar. Review the User Manual – Connectivity for establishing a network connection to the QCar. Open a terminal window in the folder where the state_estimation.py file is located:

If using a QCar 1, sudo authority is needed, run the following command:

```
sudo PYTHONPATH=$PYTHONPATH python3 state_estimation.py
```

If using a QCar 2, run the following command:

```
python state_estimation.py
```

Virtual setup -

Inside Quanser Interactive Labs navigate to the QCar Plane workspace prior to starting the skills activity. In a terminal session, navigate to the folder containing the state estimation skills activity. Note that currently a QCar 1 will be created, the spawn model for QCar 2 will be release in a future update. Use the following command to run the skills activity:

```
python state_estimation.py
```

Results for state estimation using dead reckoning:

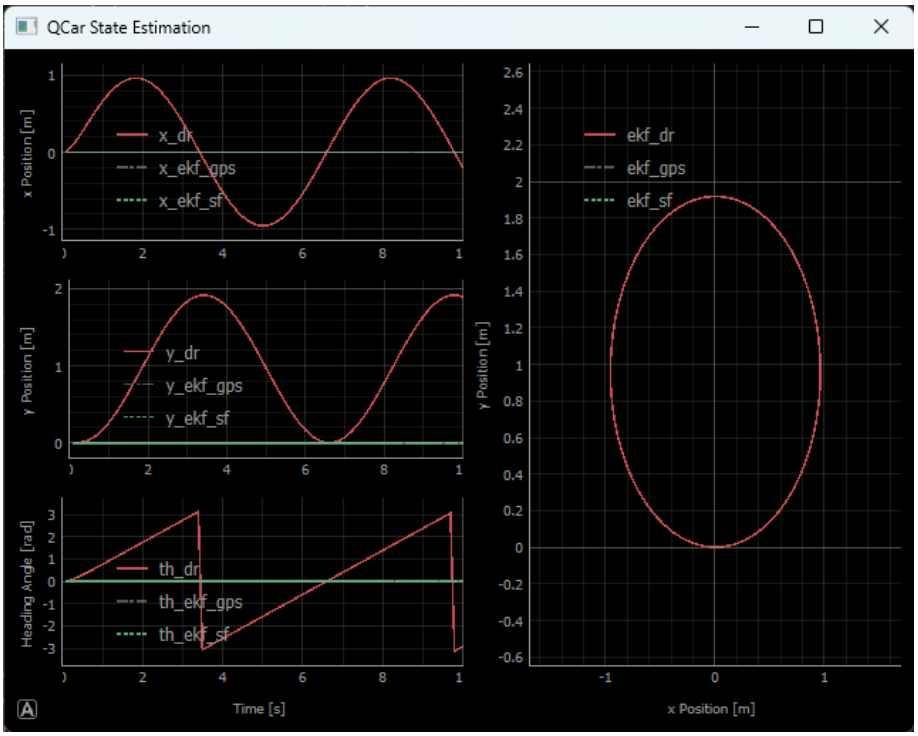

Figure 4: Results of Skills Activity 1 on Virtual QCar

Figure 4: Results of Skills Activity 1 on virtual QCar.

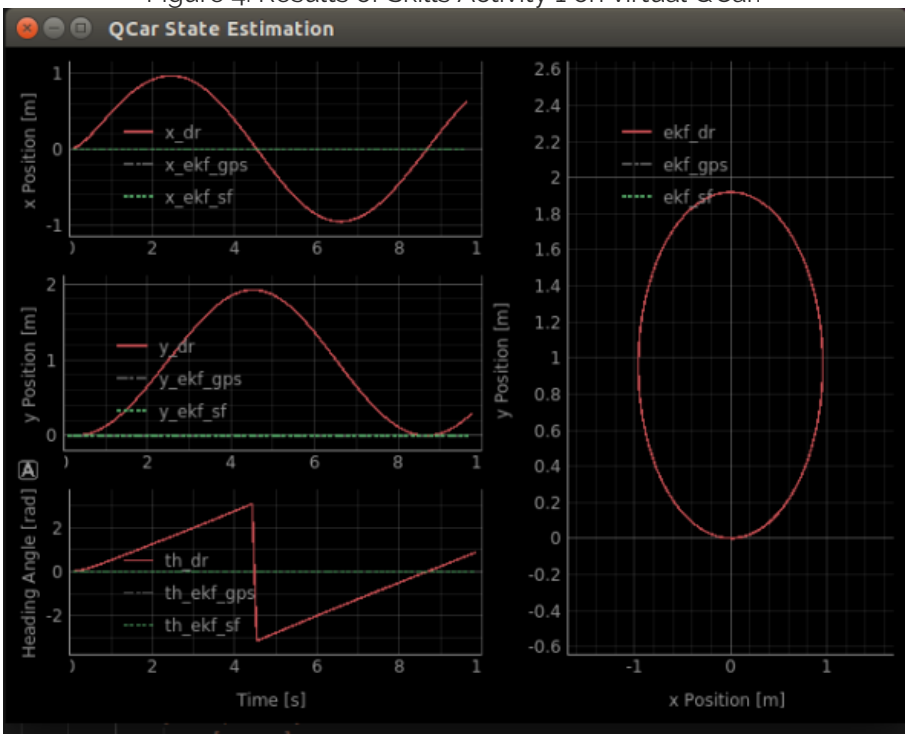

Figure 5: Results of Skills Activity 1 on Physical QCar.

Considerations

On the physical QCar make a note of the trajectory estimated using dead reckoning. How successful was the estimator at tracking the QCar as it performed a circular trajectory?

Skills Activity 2 – Extended Kalman Filter Case 1

The objective of this section is to expand the Extended Kalman Filter to include a correction step based on position information from a simulated GPS. Keep in mind the artificial GPS and the motion model are **NOT** read at the same rate. For this skill activity the GPS is read at 15Hz compared to the motion model running at the same rate as the **controllerUpdateRate**.

Note: if the skill activity will be performed on the virtual QCar, please review the Setup Guide for students on how to configure the directory structure to use **Quanser Interactive Labs**.

In this lab, you will carry out the following steps,

1. Include GPS measurement correction to the extended Kalman filter pipeline
2. Within the state estimation control loop, when to use the correction from the GPS data.

Step 1 – Calculating the State Estimate Correction based on GPS measurements

Complete **SECTION D** based on the measurement correction steps used by an extended Kalman filter. The input to the correction method is $y = [x, y, theta]$ which are the measurements from the artificial GPS.

```
    # ==============  SECTION D -  Measurement correction ====================
    def correction(self, y):
        '''
        return
```

Step 2 – Update Control Loop to include state correction

When GPS data is present it is stored in two variables:

- gps.position
- gps.orientation

Both are a [3x1] tuple with information in the following convention $[x_{axis}, y_{axis}, z_{axis}]$. In SECTION E parse out the GPS data and use the method **ekf_gps.correction(y)** to implement the correction algorithm from step 1. The variable **y** represents the state measurements the Extended Kalman Filter will correct to. This section will only run when a new GPS value is present.

```
  if gps.read():
            # ==========  SECTION E -  Measurement Update  ================

                '''
```

To enable the Extended Kalman Filter based on GPS measurements uncomment the prediction step for **ekf_gps.prediction** method in the following code block:

```
# ======= Section For enabling Kalman Filter Estimators =======
ekf_dr.prediction(dt, [speed_tach, delta])
ekf_gps.prediction(dt, [speed_tach, delta])
# ekf_sf.prediction(dt, [speed_tach, delta])
# kf.prediction(dt, th_gyro)
```

Physical setup -

Place the QCar on the floor ensuring there are no obstacles within a 1m radius of the QCar. Review the User Manual – Connectivity for establishing a network connection to the QCar. Open a terminal window in the folder where the state_estimation.py file is located:

If using a QCar 1, sudo authority is needed, run the following command:

```
sudo PYTHONPATH=$PYTHONPATH python3 state_estimation.py
```

If using a QCar 2, run the following command:

```
python state_estimation.py
```

Virtual setup -

Inside Quanser Interactive Labs navigate to the QCar Plane workspace prior to starting the skills activity. In a terminal session, navigate to the folder containing the state estimation skills activity. Use the following command to run the skills activity:

```
python state_estimation.py
```

Results for GPS based Extended Kalman Filter results:

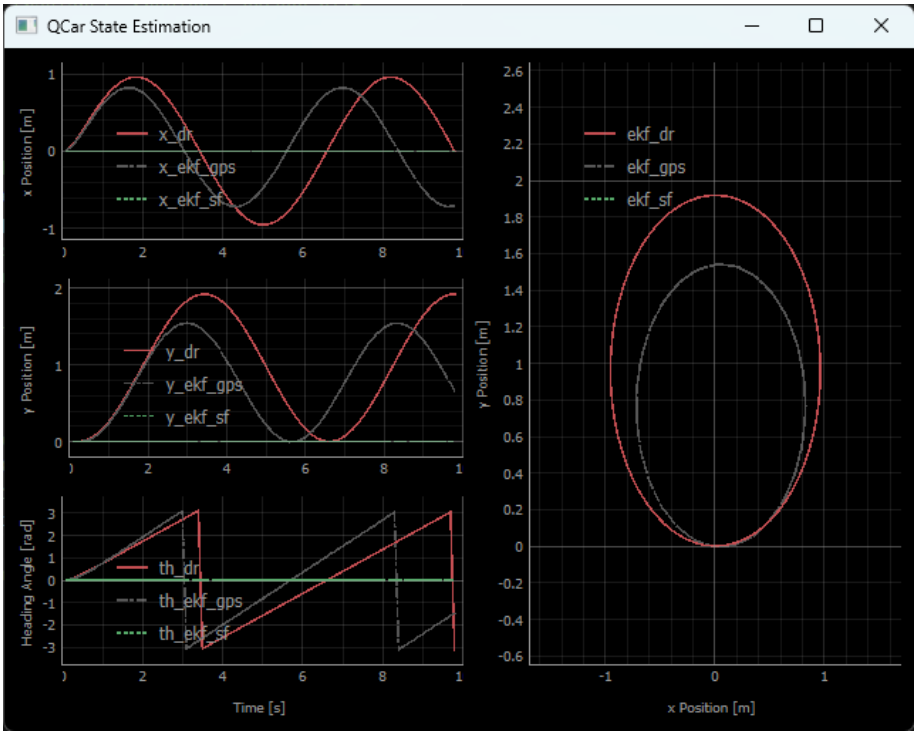

Figure 6: Results of Skills Activity 2 on virtual QCar.

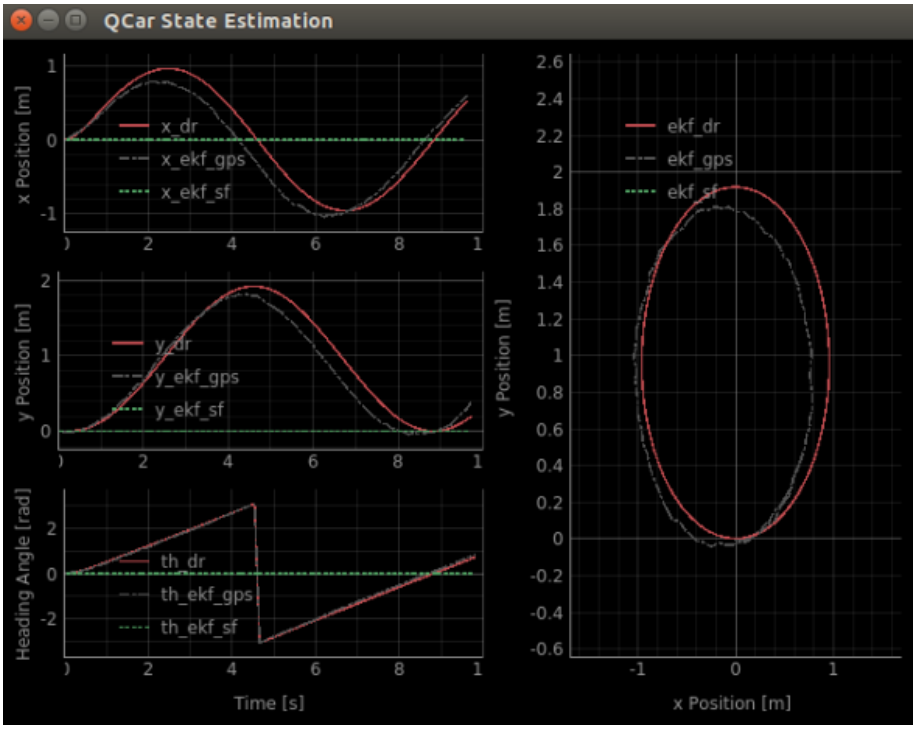

Figure 7: Results of Skills Activity 2 on physical QCar.

Considerations

How does the position estimate change from virtual to physical QCar? How does the artificial GPS measurement differ in the physical QCar?

Skills Activity 3 – Coupled Kalman Filter State Estimator

In this section the Extended Kalman Filter GPS position estimate will be augmented with a secondary filter to estimate heading. In Lab Guide – Sensor Interfacing one of the sensors looked at was the gyroscope. Gyroscopes are important sensors to consider for state estimation because they give information on the rotational rates of the system they are mounted on. For the QCar this information is useful because the z-axis measurement of the gyroscope.

In Skills Activity 2 the benefit of measurement correction was presented. However, the artificial GPS value and the controller update rate are not occurring at the same time. In this skills activity the gyroscope-z measurement (which is also occurring at the same rate as the controller update rate) will be implemented to estimate the heading of the QCar faster than what was possible in Skills Activity 2. Artificial GPS information is still used as the primary sensor to correct the entire pose estimate of the QCar.

Note: if the skill activity will be performed on the virtual QCar, please review the Setup Guide for students on how to configure the directory structure to use **Quanser Interactive Labs**.

In this lab, you will carry out the following steps,

1. Implement a Kalman Filter for estimating heading
2. Modify the Extended Kalman Filter control loop to include the estimated heading value.

Step 1 – Implementing a Kalman Filter for heading:

The state estimation skills activity includes **GyroKF** class which will be used for the heading estimation using the gyroscope and heading coming from the GPS. Building on the work done in skills activity 1 and 2 the prediction and correction steps will be implemented for the Kalman Filter.

Complete **SECTION F** to implement a discretized heading estimator based on the most recent gyroscope measurement.

```
    # ==========  SECTION F -  Gyro Heading Prediction  ================
    def prediction(self, dt, u):
        # - dt: change in time since last prediction
        # - u: most recent gyroscope measurement

        '''
```

Complete **SECTION G** to implement the prediction step based on the heading from the artificial GPS measurement.

```
# ==========  SECTION G -  GPS Heading Correction  ================
    def correction(self, y):
        # - y: heading measurement from GPS

        '''
```

Step 1 – Updating the Extended Kalman Filter control loop to include filtered heading:

Enable the sensor fusion based Kalman filter in the Estimator Section block of the control loop by uncommenting **ekf_sf.prediction** and **kf.prediction** .

```
# ======= Section For enabling Kalman Filter Estimators =======
            ekf_dr.prediction(dt, [speed_tach, delta])
            ekf_gps.prediction(dt, [speed_tach, delta])
            ekf_sf.prediction(dt, [speed_tach, delta])
            kf.prediction(dt, th_gyro)
```

In **SECTION E** include the steps for:

- Using **kf.correction(gps_heading)** which enables the correction step used by the heading Kalman Filter
- Using **ekf_sf.correction(y)** to enable the correction step of the Extended Kalman Filter which combines the Filtered heading and the artificial GPS position measurement.

```
if gps.read():
# ==========  SECTION E -  Measurement Update  ================

< ANSWER TO SKILL ACTIVITY #2 >

ekf_sf.C = C_combined
ekf_sf.R = R_combined
# Correction for estimator 3.a using GPS
'''
# Correction for estimator 3.b using GPS and heading estimate
'''


      else:
      # Correction for 3.b using only heading estimate
      '''
```

The ending of **SECTION E** is important to consider because the state estimator is combining 2 filters into a single pose estimate. Since there are two filters working at different rates, the **else** statement allows the filter to consider the estimate from the heading KF rather than the artificial GPS value which is being read at a slow rate.

Physical setup -

Place the QCar on the floor ensuring there are no obstacles within a 1m radius of the QCar. Review the User Manual – Connectivity for establishing a network connection to the QCar. Open a terminal window in the folder where the state_estimation.py file is located:

If using a QCar 1, sudo authority is needed, run the following command:

```
sudo PYTHONPATH=$PYTHONPATH python3 state_estimation.py
```

If using a QCar 2, run the following command:

```
python state_estimation.py
```

Virtual setup -

Inside Quanser Interactive Labs navigate to the QCar Plane workspace prior to starting the skills activity. In a terminal session, navigate to the folder containing the state estimation skills activity. Use the following command to run the skills activity:

```
python state_estimation.py
```

Results for Kalman Filter using Sensor Fusion:

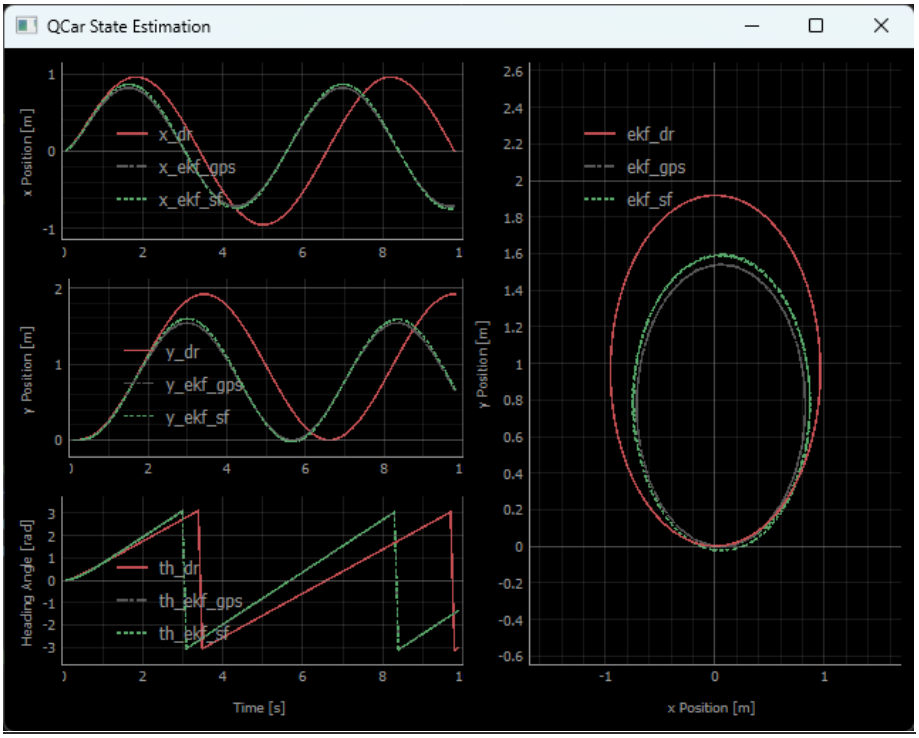

Figure 8: Results for virtual QCar

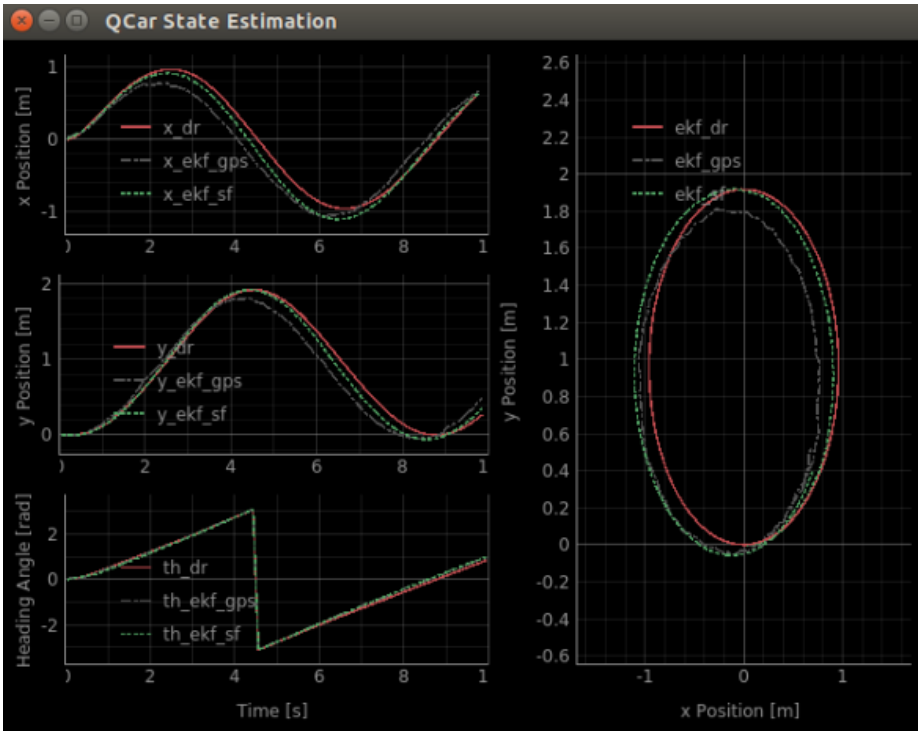

Figure 9: Results for physical QCar.

Considerations

How does the predicted pose from the fused pose estimator look like? What are some additional sensors available to improve the accuracy of the state estimator?