Pose Estimation and Object Tracking through Kalman Filter and 2D Image Recognition

Qizhan Tam[1], Eric Cristofalo[1], Zijian Wang[1]

I. Introduction

This report is intended to provide the necessary procedures and background material for understanding and to make use of the repository StanfordMSL/quad_dataset_gen[1]. There are 3 parts to this report, each denoted by **[3D: Quad]**, **[3D: Vehicle]**, or **[3D+Yaw: Vehicle]**. We first go through estimating the 3D pose of a quad in the flight room, followed by applying the same algorithm to a tram from a KITTI dataset, then lastly we introduce changes to the dynamics and measurement models to estimate the tram's yaw orientation in addition to its 3D coordinates.

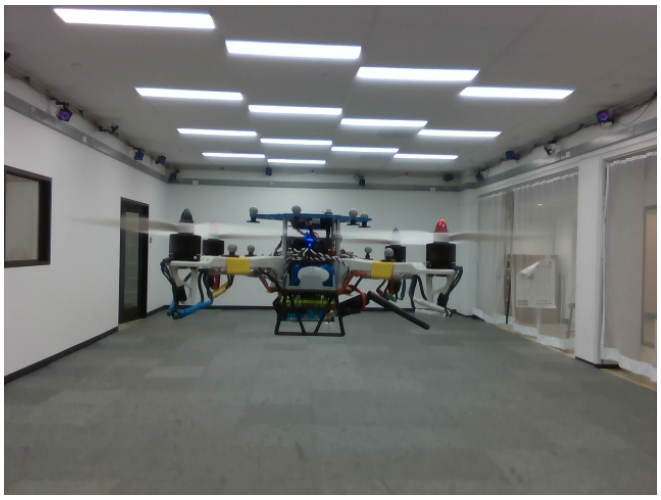
The camera used in the quad example is an Intel RealSense D435 camera, which has RGB and depth streams, which is both saved in rosbags by launching `save_data.launch` (Fig. 1).

II. [3D: Quad] Setup and Calibration

Before the start of each data collection session in the flight room, make sure to first calibrate the motion-tracking system (OptiTrack), followed by calibrating the camera's rigid body as determined by OptiTrack.

We first orient the camera such that the world (flight room) frame's x-axis is approximately pointed perpendicularly into the image (sensor) plane. After setting up a target that is the same height from the floor as the sensor, we placed the camera and the target as far apart as possible. We then used `display_image.cpp` from the StanfordMSL/msl_vision[2] repository to adjust the camera until it is in parallel to the target by matching the target to the center-line drawn on the display. This rigid body calibration will allow OptiTrack to provide us with the camera's pose in world frame, described by its homogeneous transformation matrix, $T_{\text{cam}}^{\text{world}}$, which is made up of its rotation matrix, $R_{\text{cam}}^{\text{world}}$, and its translation vector, $t_{\text{cam}}^{\text{world}}$.

We also need the coordinate transformation from the camera's frame to the image frame, $T_{\text{image}}^{\text{cam}}$. The convention for the image frame's coordinate system is having the z-axis pointed perpendicularly into the image plane. If the camera's rigid body was properly calibrated, the rotation matrix, $R_{\text{image}}^{\text{cam}}$, found in `Camera_Transformations.py` would not require changes, but $t_{\text{image}}^{\text{cam}}$ may require further fine-tuning.


(a) RGB Stream from Intel RealSense.

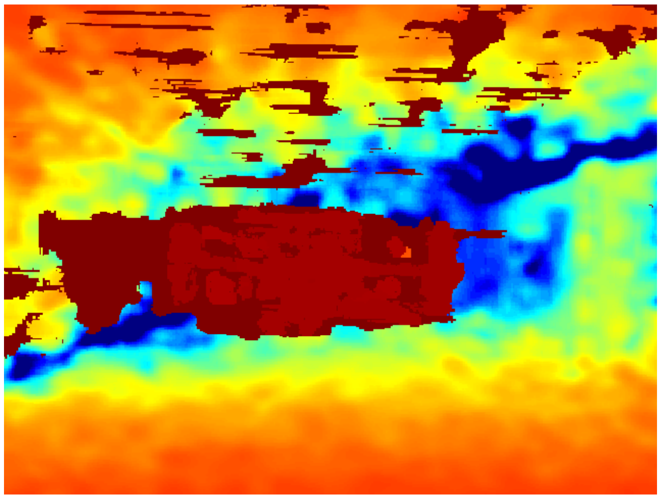

(b) Depth Stream from Intel RealSense mapped to RGB for visualization purposes.

Fig. 1: RGB stream from a monocular camera, and Depth stream from a set of stereo and IR cameras. Dataset 1: https://youtu.be/zzWaLuYzWDc; Dataset 2: https://youtu.be/PQpW8bJRn_c.

[1]Stanford University
[1]https://github.com/StanfordMSL/quad_dataset_gen
[2]https://github.com/StanfordMSL/msl_vision

III. [3D: Quad] Image Annotation for Training

There are 2 datasets collected. Dataset 1 has the quad being flown in place while the camera circles around it. Dataset 2 has the camera being stationary and the quad flying to various locations of the flightroom. To determine the 2D bounding box's parameters, we first find the quad's 3D bounding box for each image (Fig. 2a). We denote each of the bounding box vertices' coordinates as $\vec{v}^{\text{quad}}_{h,i}$, where h means that it is a homogeneous vector. Let $\vec{v}^{\text{quad}}_{h,0}$ to be the origin/center of the 3D bounding box, then for each i^{th} vertex we have:

$$\vec{v}^{\text{quad}}_{h,i} = \vec{v}^{\text{quad}}_{h,0} + \begin{bmatrix} \delta x^{\text{quad}}_i \\ \delta y^{\text{quad}}_i \\ \delta z^{\text{quad}}_i \\ 0 \end{bmatrix} \tag{1}$$

The coordinate transformation from the quad frame to the image frame in pixel coordinates can then be done as follows:

$$\vec{v}^{\text{pixel}}_{h,i} = K^{\text{pixel}}_{\text{image}} \cdot T^{\text{image}}_{\text{cam}} \cdot T^{\text{cam}}_{\text{world}} \cdot T^{\text{world}}_{\text{quad}} \cdot \vec{v}^{\text{quad}}_{h,i} \tag{2}$$

As with the camera, the quad's position and orientation are provided by OptiTrack, which is represented here as a homogeneous transformation matrix, $T^{\text{quad}}_{\text{world}} = (T^{\text{world}}_{\text{quad}})^{-1}$. We can finally obtain the 2D bounding box (Fig. 2b) coordinates ($\vec{v}^{\text{2D}}_{\text{min}} = [x_{\text{min}}, y_{\text{min}}]^{\text{T}}$, $\vec{v}^{\text{2D}}_{\text{max}} = [x_{\text{max}}, y_{\text{max}}]^{\text{T}}$) by taking the extrema of the 3D vertices, e.g., $x_{\text{min}} = \min_i x^{\text{pixel}}_i$.

Algorithm 1 Extended Kalman Filter

```
Input: $\mu_{t-1}$,$\Sigma_{t-1}$,$u_t$,$z_t$
1: $\overline{\mu}_t = g(u_t, \mu_{t-1})$
2: $\overline{\Sigma}_t = G_t \Sigma_{t-1} G_t^{\text{T}} + R_t$
3: $K_t = \overline{\Sigma}_t H_t^{\text{T}} (H_t \overline{\Sigma}_t H_t^{\text{T}} + +Q_t)^{-1}$
4: $\mu_t = \overline{\mu}_t + K_t(z_t - h(\overline{\mu}_t))$
5: $\Sigma_t = (I - K_t H_t)\Sigma_t$
6: return $\mu_t, \Sigma_t$
```

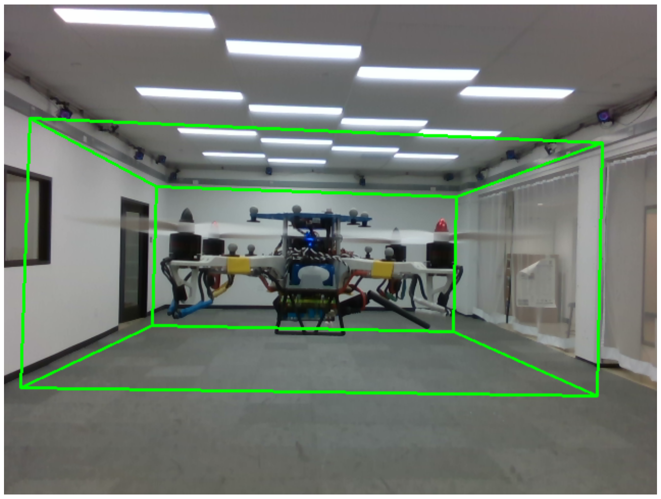
IV. [3D: Quad] Pose Estimation

We estimate the relative pose of the quad, $\vec{x}^{\text{image}} = [x^{\text{image}}, y^{\text{image}}, z^{\text{image}}]^{\text{T}}$, by using the EKF algorithm (See Algorithm 1). For the dynamics propagation step, We assume 0 relative velocity, so $\overline{\mu}_t = g(\mathbf{0}, \mu_{t-1}) = \mu_{t-1}$, and $G_t = \mathbf{I}$. As for the measurement update, we use the measurement model:

$$h(\vec{x}^{\text{image}}) = [p_x, p_y, d_x, d_y]^{\text{T}} \tag{3}$$

$$\begin{bmatrix} p_x \\ p_y \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \cdot K^{\text{pixel}}_{\text{image}} \cdot \vec{x}^{\text{image}} \tag{4}$$

$$\begin{bmatrix} d_x \\ d_y \end{bmatrix} = \begin{bmatrix} \frac{\alpha_x \cdot f \cdot w}{\sqrt{x^2+y^2+z^2}} \\ \frac{\alpha_y \cdot f \cdot h}{\sqrt{x^2+y^2+z^2}} \end{bmatrix} \tag{5}$$


(a) 3D bounding box.

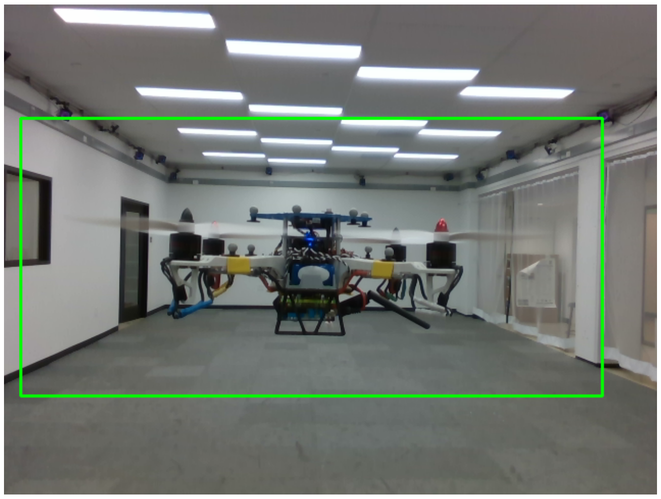

(b) 2D bounding box.

Fig. 2: The 2D bounding box is derived from the 3D bounding box by taking the extrema of the vertices. Dataset 1: https://youtu.be/55zHrJYggTE; Dataset 2: https://youtu.be/0_OVPVm0naU.

To find H, we take the Jacobian of h:

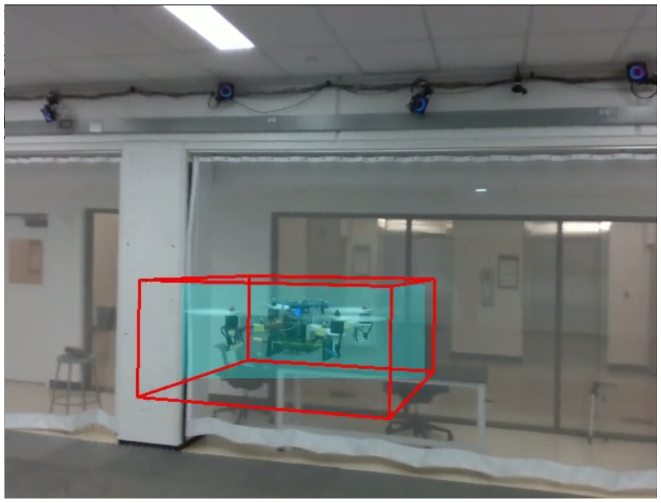
$$H(\vec{x}^{\text{image}}_{\text{quad}}) = \frac{\partial}{\partial \mathbf{x}} [p_x, p_y, d_x, d_y]^{\text{T}} \tag{6}$$

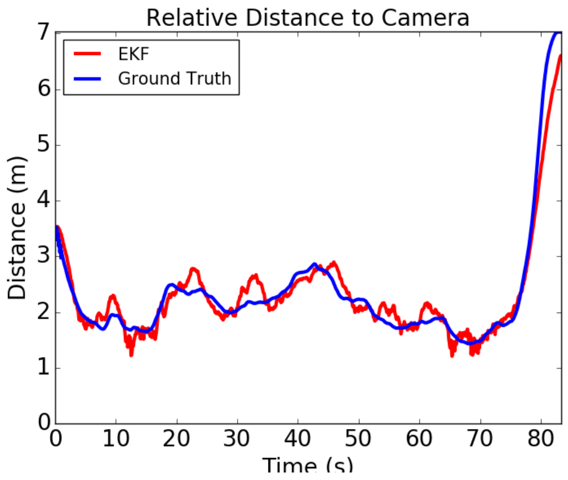
$$\frac{\partial}{\partial \mathbf{x}} \begin{bmatrix} \text{p}_x \\ \text{p}_y \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \cdot K^{\text{pixel}}_{\text{image}} \cdot \begin{bmatrix} \text{z}^{-1} & 0 & \text{-xz}^{-2} \\ 0 & 4\text{z}^{-1} & \text{-yz}^{-2} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{7}$$

$$\frac{\partial}{\partial \mathbf{x}} \begin{bmatrix} \text{d}_x \\ \text{d}_y \end{bmatrix} = \begin{bmatrix} \frac{-x\cdot\alpha_x\cdot f_x\cdot w}{(x^2+y^2+z^2)^{\frac{3}{2}}} & \frac{-y\cdot\alpha_x\cdot f_x\cdot w}{(x^2+y^2+z^2)^{\frac{3}{2}}} & \frac{-z\cdot\alpha_x\cdot f_x\cdot w}{(x^2+y^2+z^2)^{\frac{3}{2}}} \\ \frac{-x\cdot\alpha_y\cdot f_y\cdot h}{(x^2+y^2+z^2)^{\frac{3}{2}}} & \frac{-y\cdot\alpha_y\cdot f_y\cdot h}{(x^2+y^2+z^2)^{\frac{3}{2}}} & \frac{-z\cdot\alpha_y\cdot f_y\cdot h}{(x^2+y^2+z^2)^{\frac{3}{2}}} \end{bmatrix} \tag{8}$$

V. [3D: Quad] Results

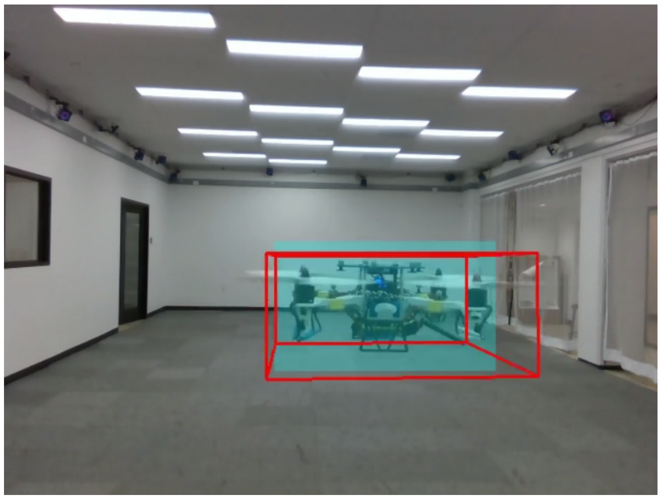
After training YOLO on Dataset 1, we use its bounding box outputs to run the EKF algorithm on both datasets. The

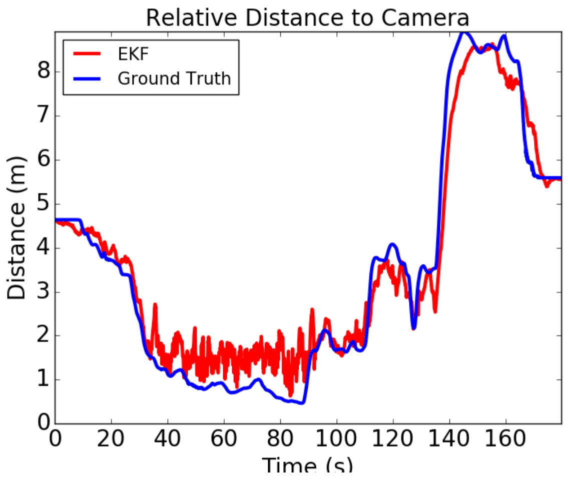

(a) Image snapshot of the estimated bounding box (red) and YOLO's detection box (cyan).



(b) Plot of the relative distance from the quad to the camera.

Fig. 3: EKF algorithm applied on Dataset 1 using bounding box output from YOLO trained on Dataset 1. Video playback: https://youtu.be/0mdw7koOrgQ.


(a) Image snapshot of the estimated bounding box (red) and YOLO's detection box (cyan).



(b) Plot of the relative distance from the quad to the camera.

Fig. 4: EKF algorithm applied on Dataset 2 using bounding box output from YOLO trained on Dataset 1. Video playback: https://youtu.be/w3OMhTWkJD8.

results for Dataset 1 can be seen in Fig. 3. For Dataset 2's results (Fig. 4), between 40-100 seconds, the pose estimations from EKF were noisy and relatively inaccurate compared to the other points in time. This is because the quad was flying much closer to the camera than it had been in the training dataset (Dataset 1), which results in inaccurate object detection.

VI. [3D: Vehicle] Setup with KITTI

This is the second part of the paper where we will track a tram from the "synced+rectified" dataset of "2011_09_26_drive_0001" provided by the KITTI Vision Benchmark Suite[3]. There are 4 video streams available to us from which we pick "cam2", one of the 2 available RGB video streams.

[3] http://www.cvlibs.net/datasets/kitti/raw_data.php

The `tracklets_label.xml` file provides us with the ground truth of the tram ($\vec{x}_h^{\text{tracklet}}$), including its dimensions. From the calibration folder, we can obtain relevant transformation matrices, i.e., $T_{\text{cam0}}^{\text{cam2}}$, $T_{\text{tracklet}}^{\text{cam0}}$, R^{rect}, $P_{\text{cam2}}^{\text{pixel}}$. If we want to find the pixel coordinates of the tram, we would compute:

$$\vec{x}_h^{\text{pixel}} = P_{\text{cam2}}^{\text{pixel}} \cdot R^{\text{rect}} \cdot T_{\text{cam0}}^{\text{cam2}} \cdot T_{\text{tracklet}}^{\text{cam0}} \cdot \vec{x}_h^{\text{tracklet}} \tag{9}$$

VII. [3D: Vehicle] Pose Estimation

For vehicle tracking, the dynamics and measurement models are very similar to the quad's, with some minor changes. We still assume that the tram has no velocity, but we are now also considering ego car's velocity:

$$u_t = [\dot{x}_{\text{ego}}, \dot{y}_{\text{ego}}, \dot{z}_{\text{ego}}]^{\text{T}} \tag{10}$$

$$g(u_t, \vec{x}_{t-1}) = \vec{x}_{t-1}^{\text{cam2}} - u_t \cdot \text{d}t \tag{11}$$

As the tram has a rectangular shape unlike the quad's square shape, we only consider the pixel height for the measurement model:

$$h(\vec{x}^{\text{image}}) = [p_x, p_y, d_y]^{\text{T}} \tag{12}$$

$$\begin{bmatrix} \text{p}_x \\ \text{p}_y \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \cdot P^{\text{pixel}}_{\text{cam2}} \cdot R^{\text{rect}} \cdot \vec{x}^{\text{cam2}} \tag{13}$$

$$d_y = \frac{\alpha_y \cdot f \cdot h}{\sqrt{x^2 + y^2 + z^2}} \tag{14}$$

To find H, we take the Jacobian of h:

$$H(\vec{x}^{\text{image}}) = \frac{\partial}{\partial \mathbf{x}} [p_x, p_y, d_y]^{\text{T}} \tag{15}$$

$$\frac{\partial}{\partial \mathbf{x}} \begin{bmatrix} \text{p}_x \\ \text{p}_y \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \cdot P^{\text{pixel}}_{\text{cam2}} \cdot R^{\text{rect}} \cdot \begin{bmatrix} \text{z}^{-1} & 0 & \text{-xz}^{-2} \\ 0 & 4\text{z}^{-1} & \text{-yz}^{-2} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{16}$$

$$\frac{\partial d_y}{\partial \mathbf{x}} = \left[\frac{-x \cdot \alpha_y \cdot f_y \cdot h}{(x^2+y^2+z^2)^{\frac{3}{2}}}, \quad \frac{-y \cdot \alpha_y \cdot f_y \cdot h}{(x^2+y^2+z^2)^{\frac{3}{2}}}, \quad \frac{-z \cdot \alpha_y \cdot f_y \cdot h}{(x^2+y^2+z^2)^{\frac{3}{2}}} \right] \tag{17}$$

VIII. [3D: Vehicle] Results

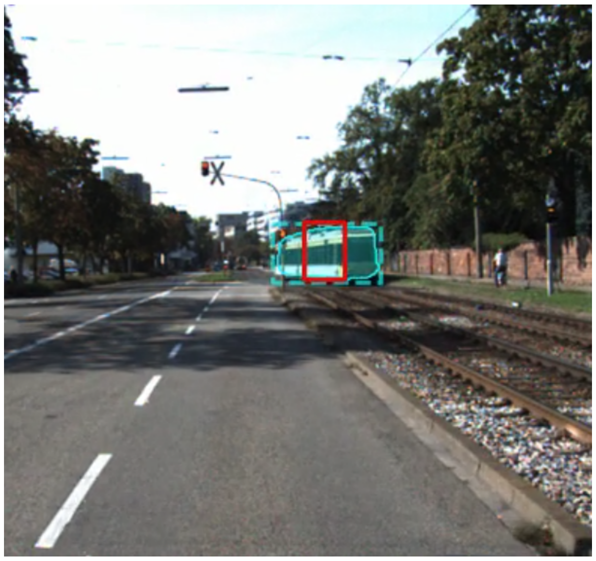
From the results shown in Fig. 5, the relative yaw angle begins to increasingly deviate away from the ground truth from 2-6 seconds. This is because while the tram was turning at an angle, the measurement model could not discern the location of the tram's rear section which the model is supposed to be tracking. The noticeable drift in both relative distance and yaw angle between 7-10 seconds is due to the lack of measurement readings from Mask-RCNN. Without regular readings, the tram position remains constant. This highlights the downsides of setting the tram to have 0 velocity.

IX. [3D+Yaw: Vehicle] Setup 2.0

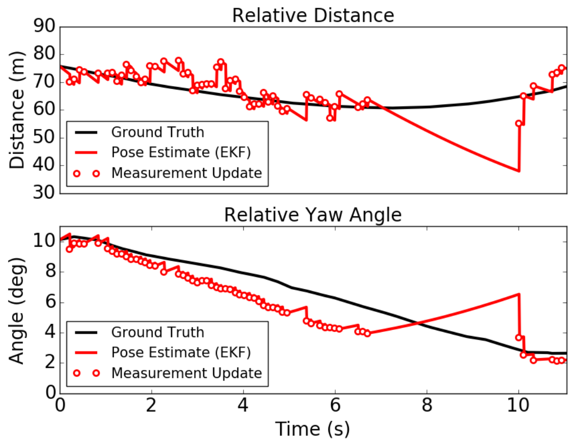
Due to the tram's rectangular form, we can estimate its yaw orientation by utilizing the pixel width which varies with its yaw. We take a different approach in formulating the measurement model, this time directly projecting the relevant vertices into the pixel frame.

Illustrated in Fig. 6, there are 3 different sets of vertices that need to be projected depending on which parts are visible to the camera. The first set (Case 1) is relevant when only the back vertices are visible, the second set (Case 2) is when the left side of the front vertices are visible, and the third set (Case 3) is when the right side of the front vertices are visible. At each time-step, all vertices are calculated by using the first 4 components of the pose estimate, $\vec{x}^{\text{cam2}} = [x^{\text{cam2}}, y^{\text{cam2}}, z^{\text{cam2}}, \theta^{\text{cam2}}, V^{\text{cam2}}, \omega^{\text{cam2}}]^{\text{T}}$:

$$\vec{v}^{\text{cam2}}_i = \vec{v}^{\text{cam2}} + \vec{\delta v}_i(\theta^{\text{cam2}}) \tag{18}$$



(a) Image snapshot of the estimated 2D bounding box (red) and the Mask-RCNN's detection bounding box (blue).



(b) Plots of the relative distance and the relative angle from the ego car to the tram.

Fig. 5: EKF algorithm applied to KITTI dataset for 3D pose estimation. Video playback of results: https://youtu.be/fkqxKGVjbVE.

X. [3D+Yaw: Vehicle] Pose Estimation 2.0

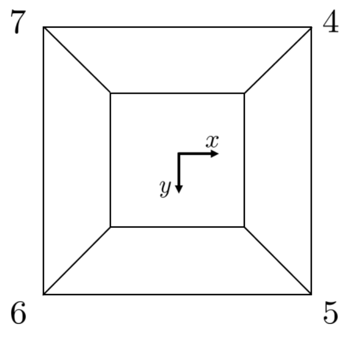
For dynamics, we use the Dubins car model with constant velocity:

$$u_t = \begin{bmatrix} -V \cdot \sin\theta - \dot{x}_{\text{ego}} \\ -\dot{y}_{\text{ego}} \\ V \cdot \sin\theta - \dot{z}_{\text{ego}} \\ \omega - \dot{\theta}_{\text{ego}} \\ 0 \end{bmatrix} \tag{19}$$

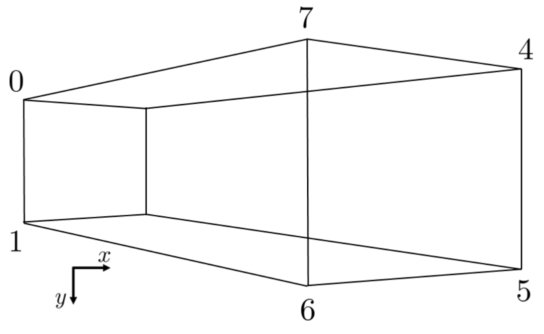
$$g(u_t, \vec{x}^{\text{cam2}}_{t-1}) = \vec{x}^{\text{cam2}}_{t-1} + u_t \cdot \text{dt} \tag{20}$$

with $G = \mathbf{J}_g$. As for the measurement model, we first project the vertices from the camera frame to the pixel frame,

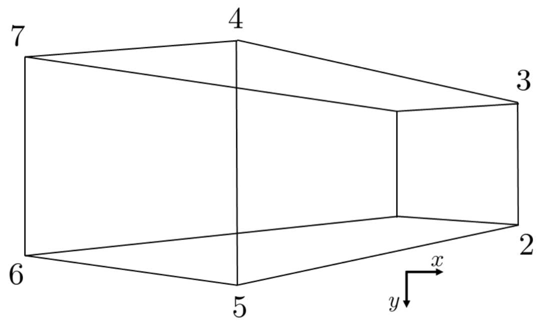
$$\vec{v}^{\text{pixel}}_{h,i} = P^{\text{pixel}}_{\text{cam2}} \cdot R^{\text{rect}} \cdot \vec{v}^{\text{cam2}}_{h,i} \tag{21}$$



(a) Case 1 ($c = 1$): Center Orientation

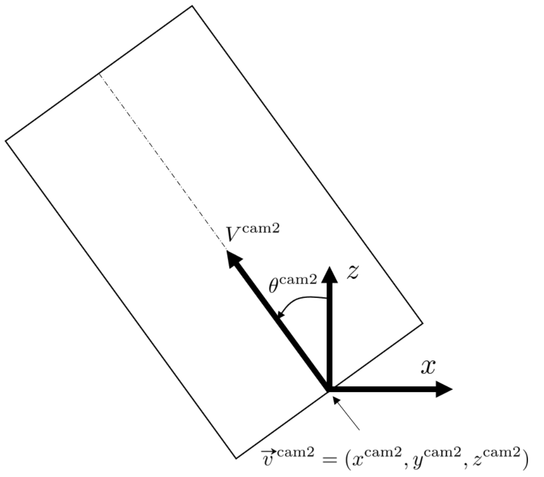


(b) Case 2 ($c = 2$): Left Orientation



(c) Case 3 ($c = 3$): Right Orientation

Fig. 6: Different projections that lead to discrete cases of measurement models.



V^{cam2}
θ^{cam2}
z
x
$\vec{v}^{\text{cam2}} = (x^{\text{cam2}}, y^{\text{cam2}}, z^{\text{cam2}})$

Fig. 7: Elements of the pose vector that we are estimating.

then calculate the expected center and width of the bounding box. For example, for Case 1 ($c = 1$), when only the back vertices are visible:

$$h_c(\vec{v}_{h,i}^{\text{pixel}} : i = \{4, 5, 6\}) = [p_x, p_y, d_x, d_y]^{\text{T}} \tag{22}$$

$$\begin{bmatrix} \text{p}_x \\ \text{p}_y \\ \text{d}_x \\ \text{d}_y \end{bmatrix} = \begin{bmatrix} (x_5 + x_6)/2 \\ (y_5 + y_4)/2 \\ x_5 - x_6 \\ y_5 - y_4 \end{bmatrix} \tag{23}$$

As there are 3 possible cases, $c \in \{1, 2, 3\}$, we need a method to choose the case, c^*, that best aligns with reality for the measurement update. We use $K_t(z_t - h(\overline{\mu}_t))$ from line 4 of Algorithm 1 as a measure of model mismatch, where $\overline{\mu}_t$ is the propagation of the previous estimated state using the prescribed dynamics. Hence, for the measurement update, we use the model, c^*, from:

$$c^* = \underset{c}{\text{argmin}} \; \|K_t(z_t - h_c(\overline{\mu}_t))\|_{\text{F}} \tag{24}$$

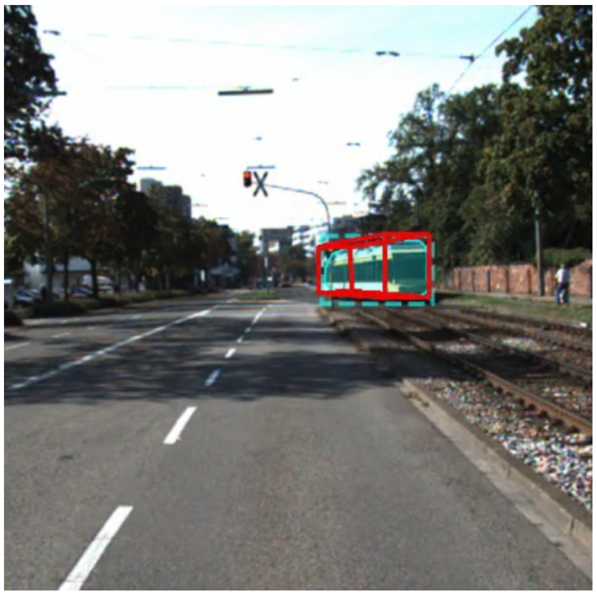
For example, a tram angled left could have the same bounding box dimensions as a tram angled right. If the tram is travelling towards the left, the prediction step in line 1 will produce a state, when used with the Case 2 (angled left) measurement model, will have the best agreement with the actual measurement, z_t. For a more robust approach, we could run parallel cases for a finite time horizon, but a 1-timestep evaluation was sufficient for this example.

XI. [3D+Yaw: Vehicle] Results

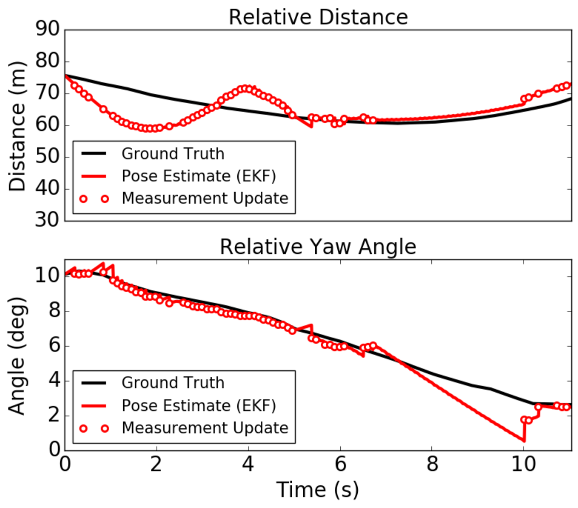
The addition of the Dubins Car dynamics model and the updated measurement models resulted in more continuous pose updates, which is reflected in Fig. 8, most notably between 0-5 seconds in the relative distance plot. The estimated relative yaw angle was also able to track the ground truth significantly better with minimal deviations except for the period between 7-10 seconds. The lack of measurement updates between 7-10 seconds still affected the tracking of states, but the propagation of states by the dynamics model greatly reduced the tracking error. The model mismatch metric was also able to determine the correct measurement model to use, except during one update at 10s as it was immediately after a notable absence of measurement updates.

Acknowledgement

I would like to thank Prof. Schwager for giving me this opportunity to pursue this research and providing me with important ideas during our discussions. Eric and Zijian have also been great mentors to me, as I would not have been able to get nearly as much done without their help and advice.



(a) Image snapshot of the estimated 3D bounding box (red) and the Mask-RCNN's detection bounding box (blue).



(b) Plots of the relative distance and the relative angle from the ego car to the tram.

Fig. 8: EKF algorithm applied to KITTI dataset with Dubins Car model and discrete cases of measurement models. Video playback of results: https://youtu.be/rPjyuLXU-7w.