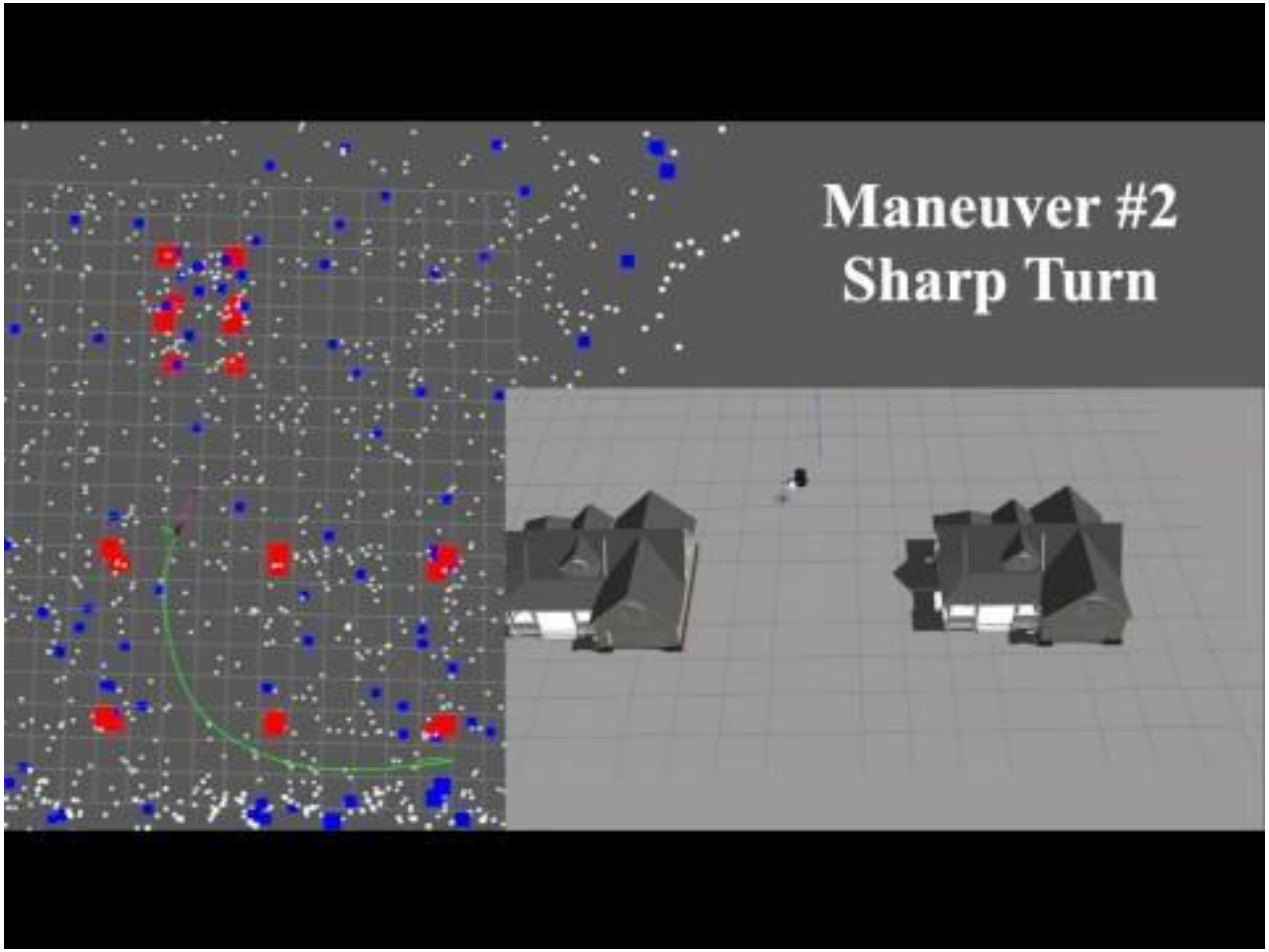
Kalman Filtering for Visual-Inertial Navigation and Target Tracking

Kevin Eckenhoff

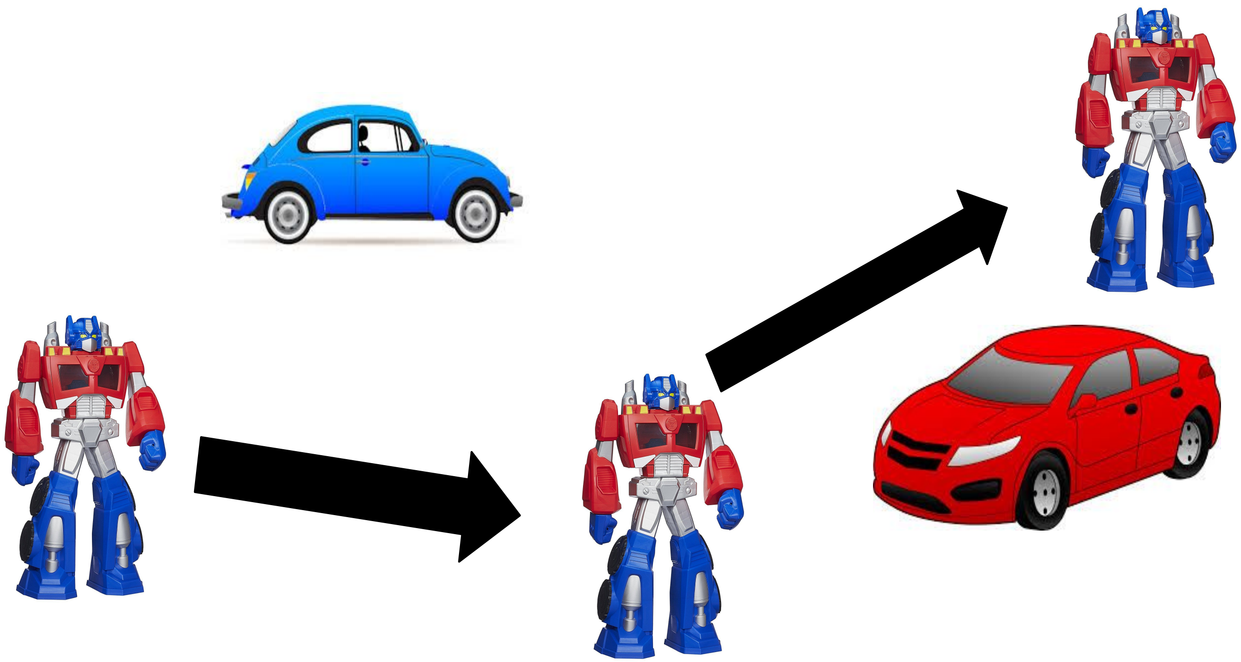
Active Target Tracking



Navigation for Robotics

- Navigation: want to track the motion of robot moving through some unknown environment using only onboard sensors

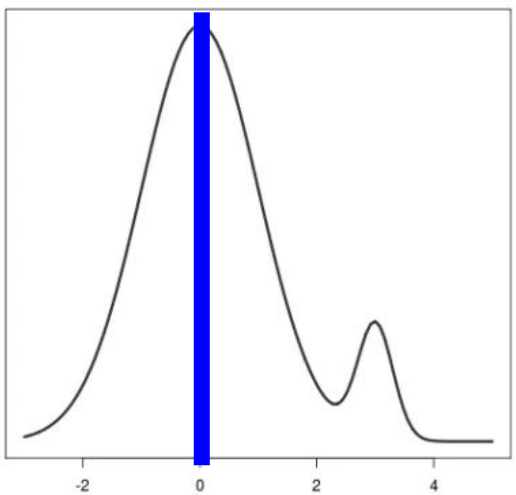
- Applications: many (e.g., search and rescue, extraplanetary exploration, autonomous driving, defeating the Decepticons)



Estimation

- Consider an unknown random variable, $\mathbf{x}$, and a set of measurements that relate to this state, $\mathbf{z}$
- Maximum a Posteriori (MAP) estimation involves finding the most likely value of the state given the measurements

$$\hat{\mathbf{x}} = \arg\max_{\mathbf{x}} p\left(\mathbf{x} \middle| \mathbf{z}\right)$$

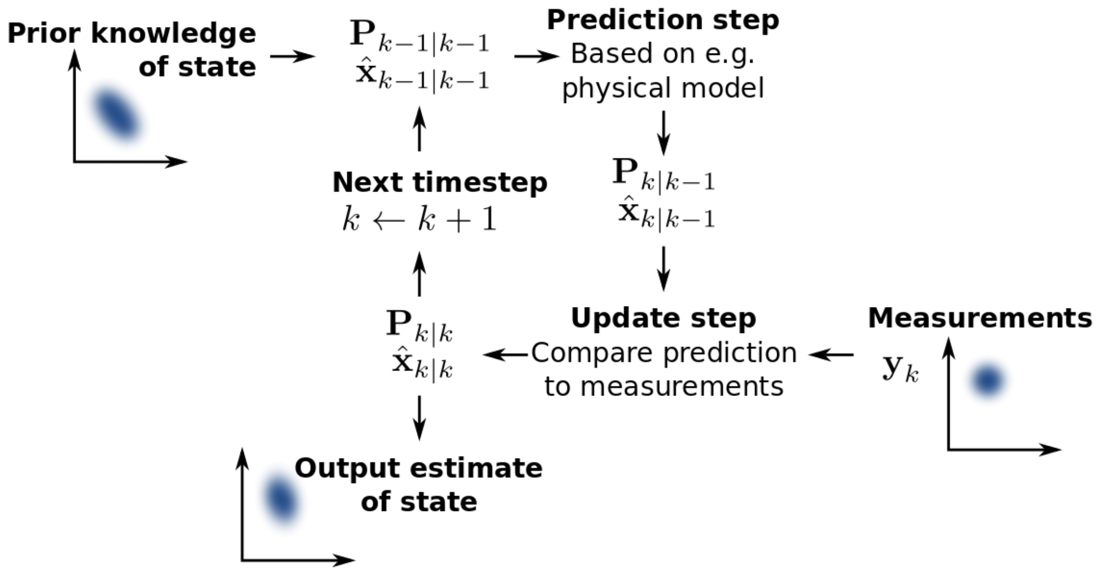


The Kalman Filter

- One of the most popular algorithms due to simplicity and low computational complexity is the Kalman Filter

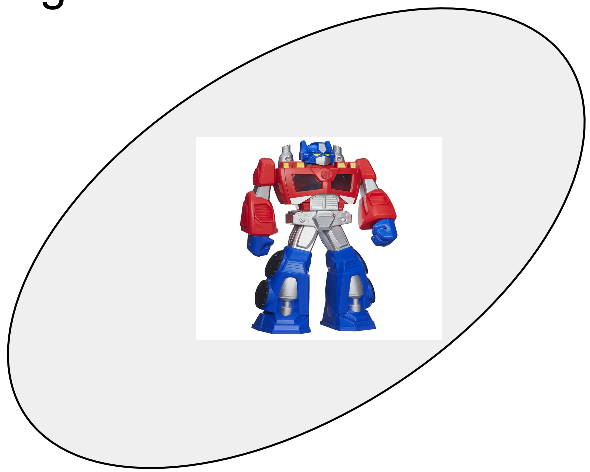
- Key idea of the algorithm: divide the algorithm into a two-step process

- Propagation predicts the next estimate of the variable based on its dynamics, while the second step updates this estimate using other sensor measurements



Gaussian Representation

- Represent the state distribution as a Gaussian, which yields a closed form expression for the MAP estimate
- The mean represents the most likely value for our random variable given the measurements, while the covariance encodes our uncertainty about this estimate
- “Optimus Prime is *somewhere near* the mean with some uncertainty”
- At every time instance we approximate the state’s distribution as a Gaussian and track the resulting mean and covariance



$$\mathbf{x}_0 \sim \mathcal{N}\left(\hat{\mathbf{x}}_{0|0}, \mathbf{P}_{0|0}\right)$$

Propagation

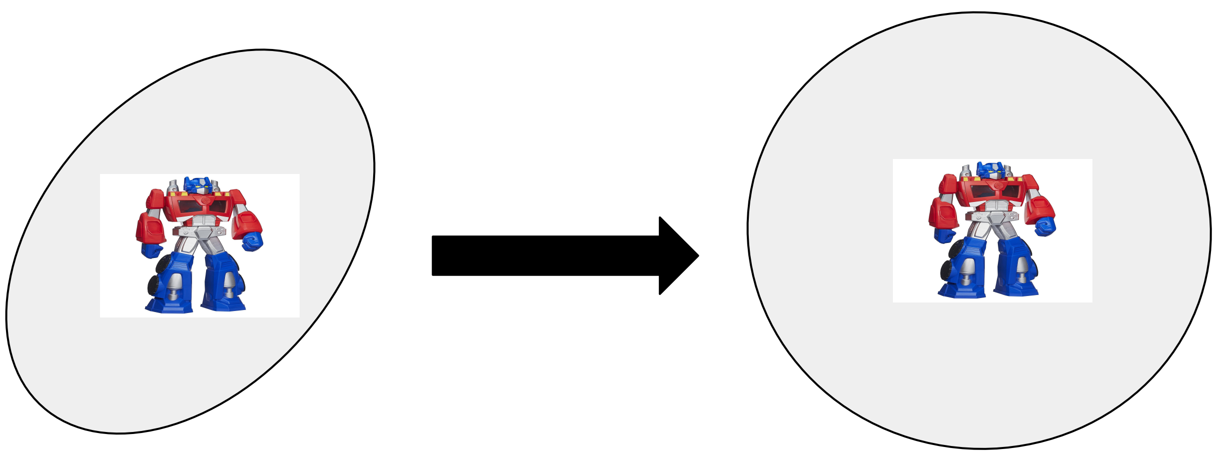
- State evolves according to some known motion model, which is a function of the state, measured inputs, and some noise

$$\mathbf{x}_1 = \mathbf{f}\left(\mathbf{x}_0, \mathbf{u}_0, \mathbf{n}_0\right)$$

- Propagation allows us to predict the state at the next instance while capturing the new covariance

- “Optimus uses noisy odometry to predict his next state with increased uncertainty”

$$\mathbf{x}_0 \sim \mathcal{N}\left(\hat{\mathbf{x}}_{0|0}, \mathbf{P}_{0|0}\right) \qquad \mathbf{x}_1 \sim \mathcal{N}\left(\hat{\mathbf{x}}_{1|0}, \mathbf{P}_{1|0}\right)$$

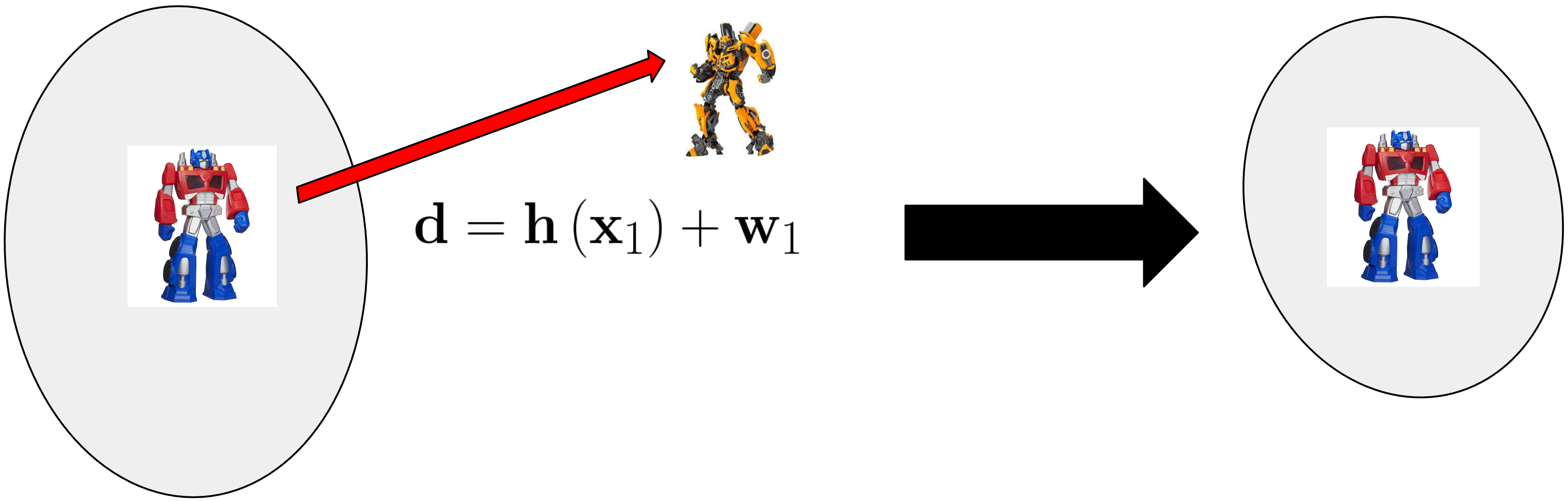


Update

- Update allows us to refine the current estimate using other sensor measurements, which are also corrupted by noise

- “Optimus Prime measures the distance from himself to the known location of Bumblebee, allowing him to update his estimate and reduce his uncertainty”

$$\mathbf{x}_1 \sim \mathcal{N}\left(\hat{\mathbf{x}}_{1|0}, \mathbf{P}_{1|0}\right) \qquad \mathbf{x}_1 \sim \mathcal{N}\left(\hat{\mathbf{x}}_{1|1}, \mathbf{P}_{1|1}\right)$$

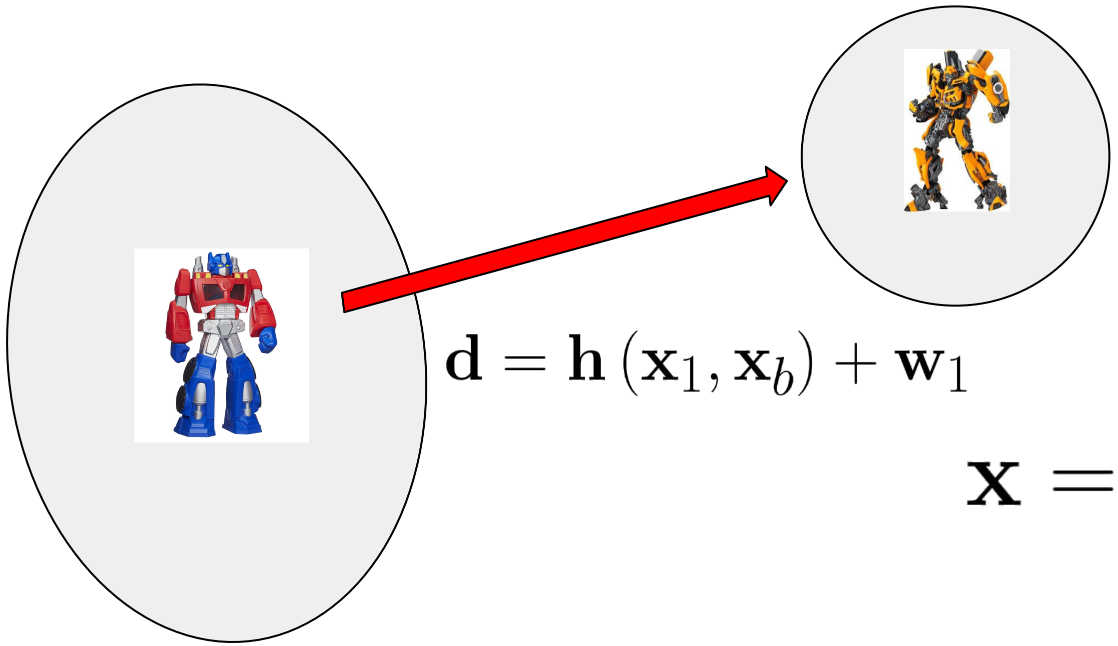


$$\mathbf{d} = \mathbf{h}\left(\mathbf{x}_1\right) + \mathbf{w}_1$$

SLAM

- If the measurement is a function of another unknown, we must add that unknown to our filter's state. When this unknown is a static landmark, this becomes the simultaneous localization and mapping (SLAM) problem

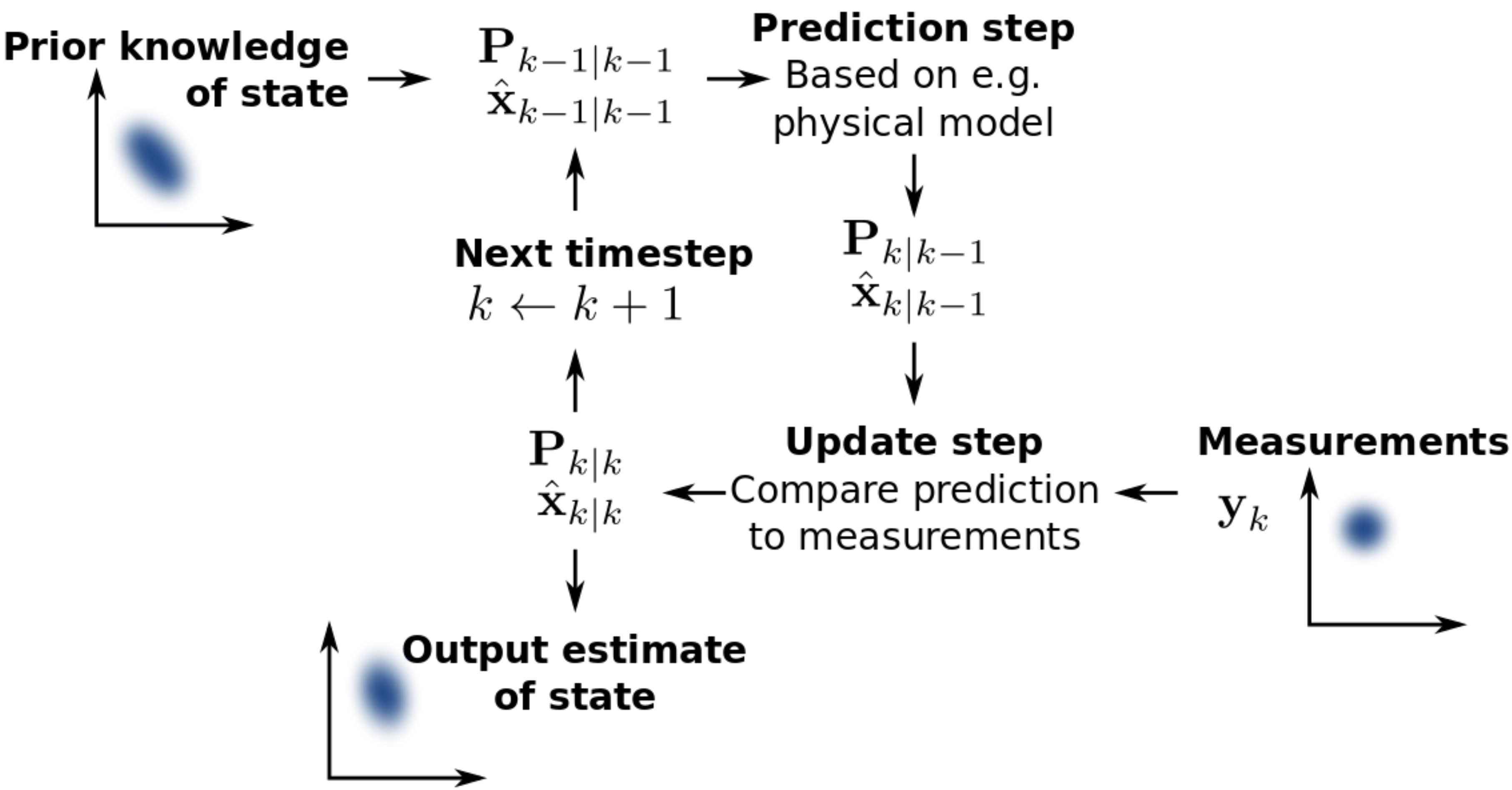
- "Optimus Prime estimates both his state and that of Bumblebee"



$$\mathbf{d} = \mathbf{h}\left(\mathbf{x}_1, \mathbf{x}_b\right) + \mathbf{w}_1$$

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_o \\ \mathbf{x}_b \end{bmatrix}$$

Kalman Filter Process



Kalman Filter Algorithm

Predict [edit]

Predicted state estimate	$\hat{\boldsymbol{x}}_{k\mid k-1} = f(\hat{\boldsymbol{x}}_{k-1\mid k-1}, \boldsymbol{u}_k)$
Predicted covariance estimate	$\boldsymbol{P}_{k\mid k-1} = \boldsymbol{F}_k \boldsymbol{P}_{k-1\mid k-1} \boldsymbol{F}_k^\top + \boldsymbol{Q}_k$

Update [edit]

Innovation or measurement residual	$\tilde{\boldsymbol{y}}_k = \boldsymbol{z}_k - h(\hat{\boldsymbol{x}}_{k\mid k-1})$
Innovation (or residual) covariance	$\boldsymbol{S}_k = \boldsymbol{H}_k \boldsymbol{P}_{k\mid k-1} \boldsymbol{H}_k^\top + \boldsymbol{R}_k$
Near-optimal Kalman gain	$\boldsymbol{K}_k = \boldsymbol{P}_{k\mid k-1} \boldsymbol{H}_k^\top \boldsymbol{S}_k^{-1}$
Updated state estimate	$\hat{\boldsymbol{x}}_{k\mid k} = \hat{\boldsymbol{x}}_{k\mid k-1} + \boldsymbol{K}_k \tilde{\boldsymbol{y}}_k$
Updated covariance estimate	$\boldsymbol{P}_{k\mid k} = (\boldsymbol{I} - \boldsymbol{K}_k \boldsymbol{H}_k) \boldsymbol{P}_{k\mid k-1}$

where the state transition and observation matrices are defined to be the following Jacobians

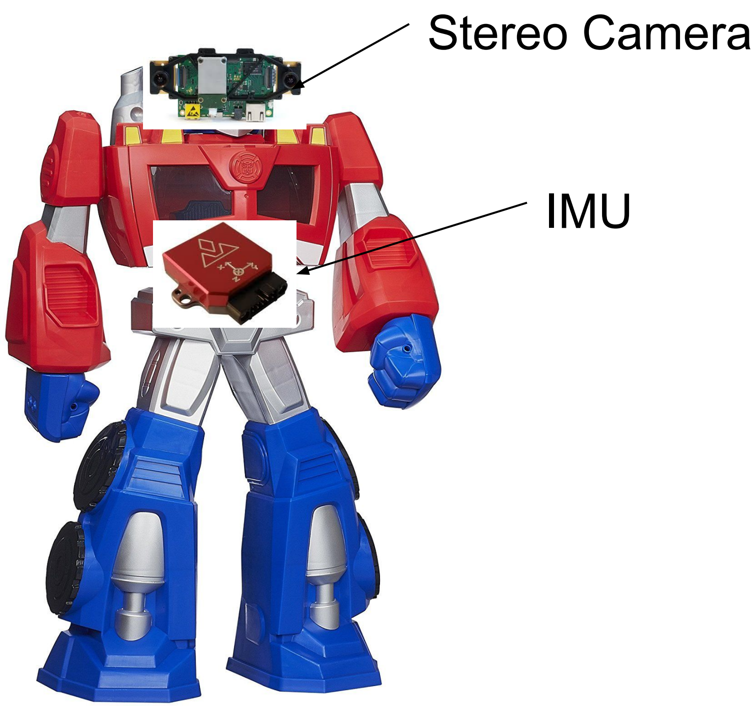
$$\boldsymbol{F}_k = \left.\frac{\partial f}{\partial \boldsymbol{x}}\right|_{\hat{\boldsymbol{x}}_{k-1\mid k-1}, \boldsymbol{u}_k}$$

$$\boldsymbol{H}_k = \left.\frac{\partial h}{\partial \boldsymbol{x}}\right|_{\hat{\boldsymbol{x}}_{k\mid k-1}}$$

Visual-Inertial Navigation

- Visual-Inertial Navigation: providing estimates for a system using an inertial measurement unit and one or more cameras.

- Kalman filtering remains an extremely popular solution to this problem due to its low computational overhead compared to other methods



Inertial Measurement Unit (IMU)

- IMU provides local linear acceleration and angular velocity measurements corrupted by both Gaussian noise and time-varying biases

- Solving resulting ODEs allows us to perform propagation

$$\mathbf{x} = \begin{bmatrix} {}^{L}_{G}q \\ {}^{G}\mathbf{p}_{L} \\ {}^{G}\mathbf{v}_{L} \\ \mathbf{b}_{w} \\ \mathbf{b}_{a} \end{bmatrix}$$

$$\mathbf{a}_m = {}^{L}\mathbf{a} + {}^{L}_{G}\mathbf{R}\,{}^{G}\mathbf{g} + \mathbf{b}_a + \mathbf{n}_a$$

$$\mathbf{w}_m = {}^{L}\mathbf{w} + \mathbf{b}_w + \mathbf{n}_w$$

$${}^{G}\dot{\mathbf{v}}_{L} = {}^{G}_{L}\mathbf{R}\,{}^{L}\mathbf{a}$$

$${}^{G}\dot{\mathbf{p}}_{L} = {}^{G}\mathbf{v}_{L}$$

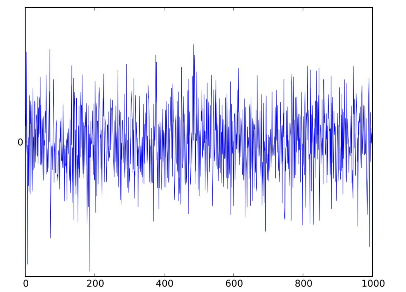
$${}^{L}_{G}\dot{\mathbf{R}} = -\lfloor {}^{L}\mathbf{w} \rfloor\, {}^{L}_{G}\mathbf{R}$$

IMU Biases

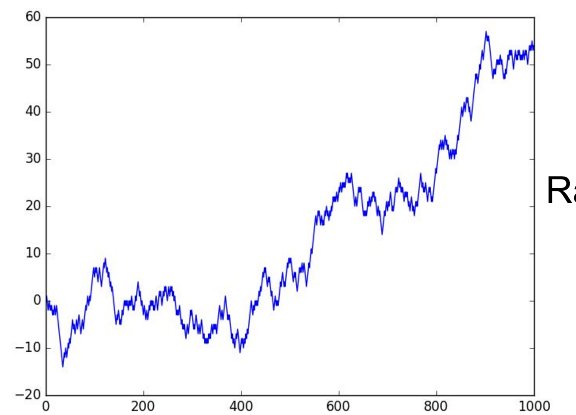
- Biases are modeled as continuous-time random walks driven by Gaussian noise

- These model effects which cannot be captured by pure Gaussian noise

$$\dot{\mathbf{b}}_w = \mathbf{n}_{bw}$$
$$\dot{\mathbf{b}}_a = \mathbf{n}_{ba}$$



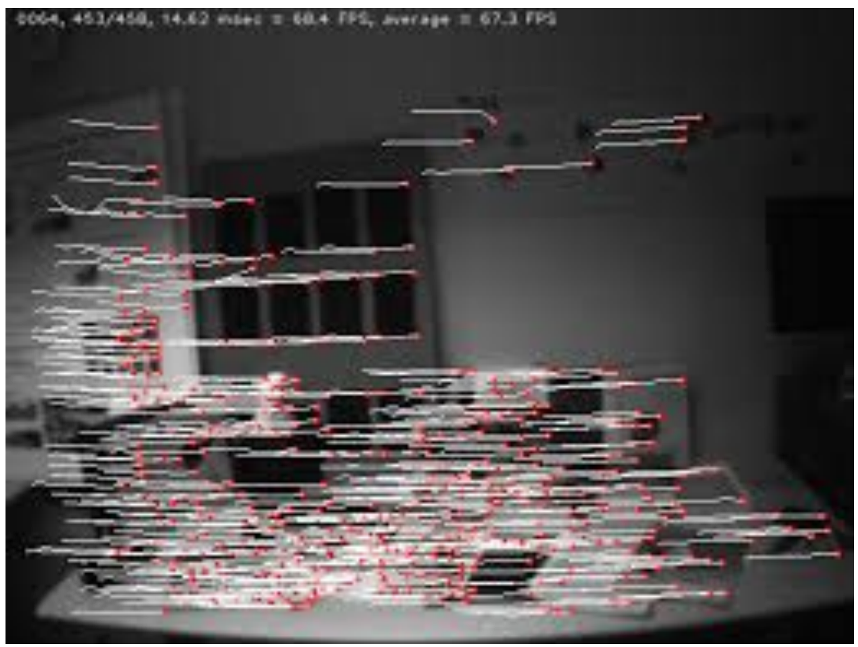
White Noise



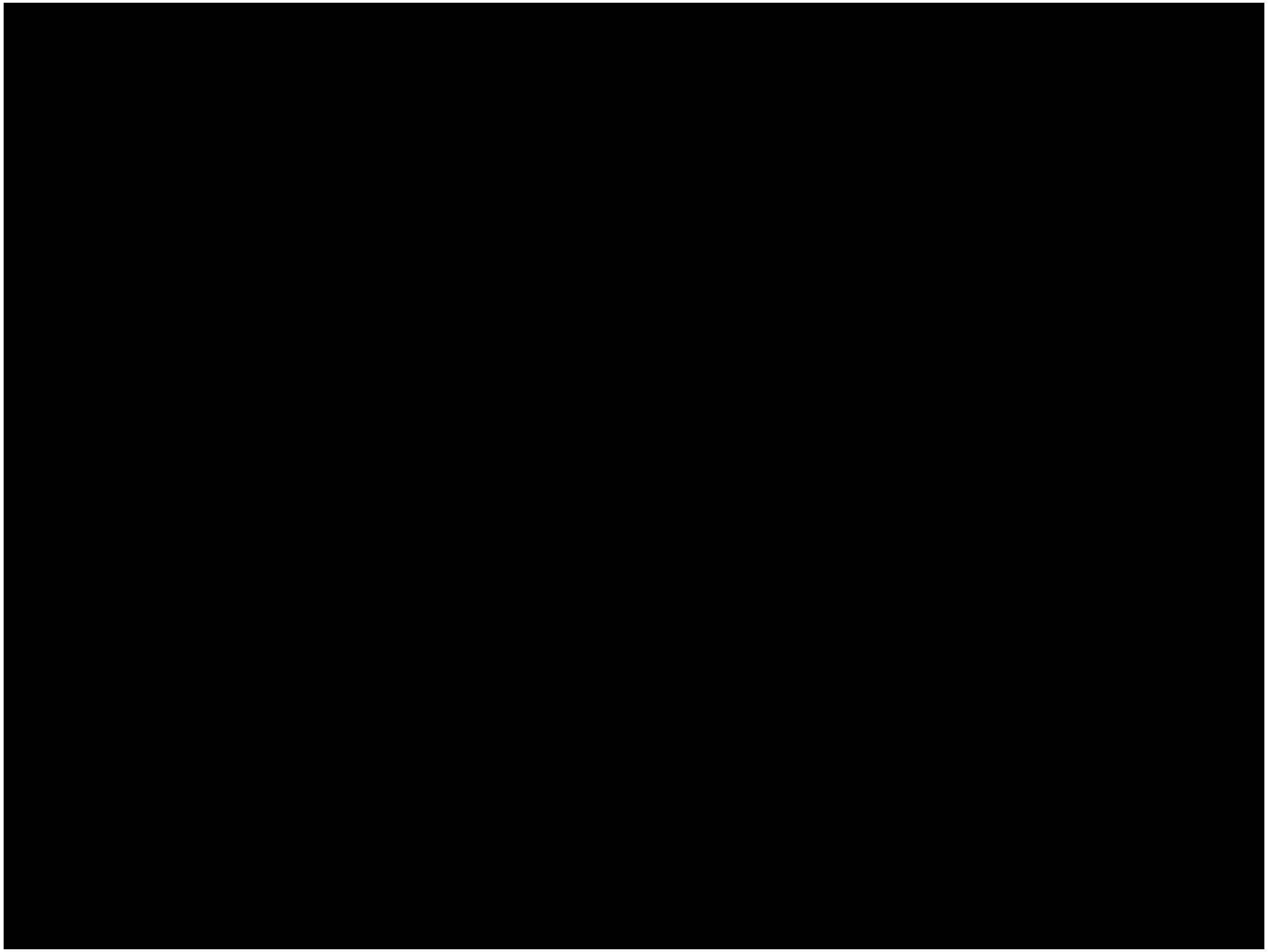
Random Walk

Camera Measurements

- Cameras capture images of the unknown, surrounding environment
- Standard image processing allows us to extract and track features across a series of images, which provide us with information of both the environment and the motion of the sensor



KLT Tracking



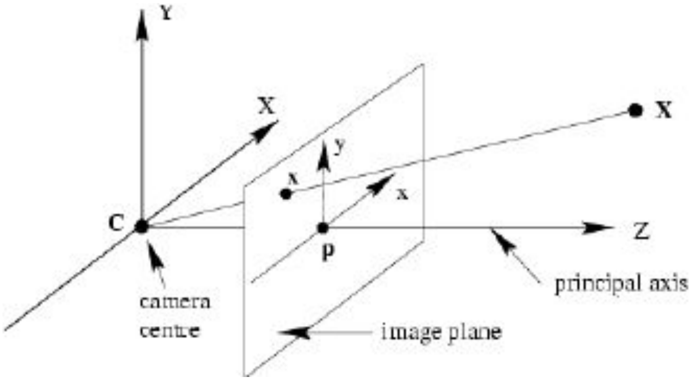
Projection Function

- Features in the 2D image correspond to static 3D points in the environment (e.g., a corner of a table) with position ${}^{G}\mathbf{p}_f$.

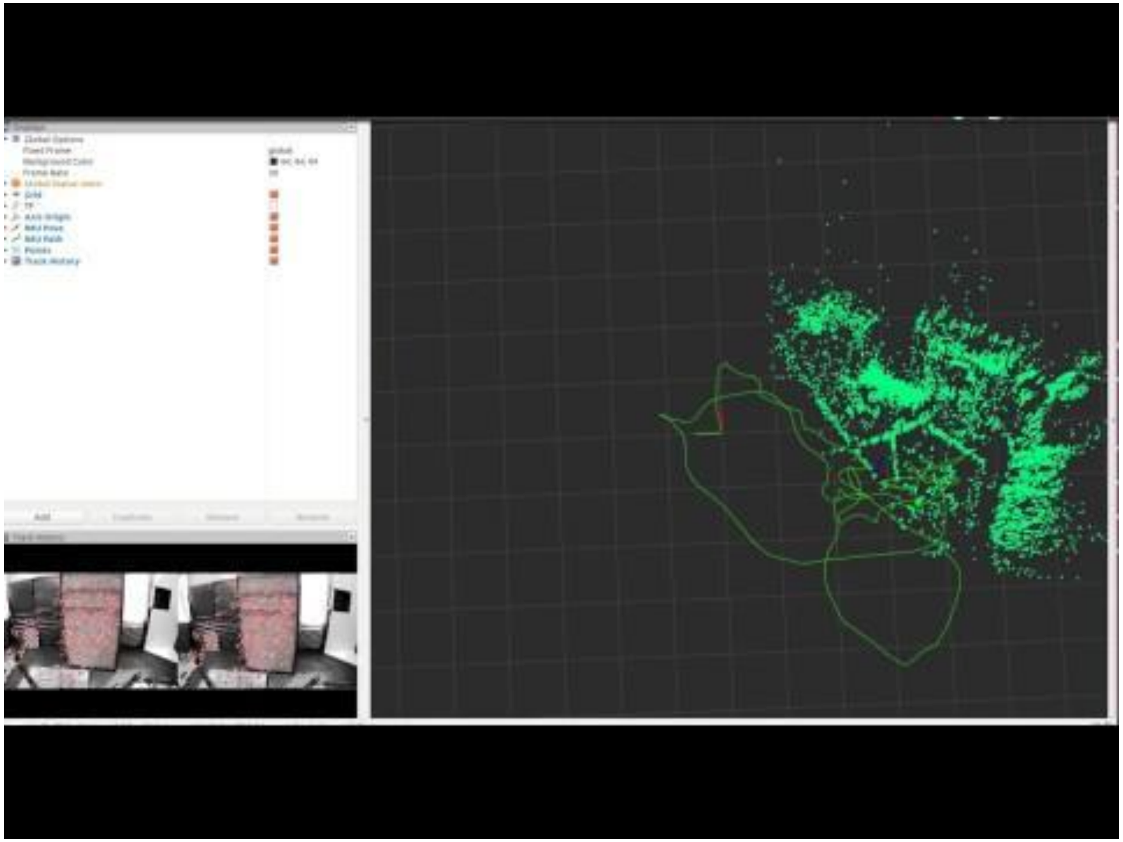
- Image coordinate measurements are related to the sensor state and feature position by a projection function

$$\mathbf{z} = \mathbf{h}\left(\mathbf{x}, {}^{G}\mathbf{p}_f\right) + \mathbf{w}$$

- This function first transforms the 3D point into the camera frame of reference, followed by a projection onto the image plane

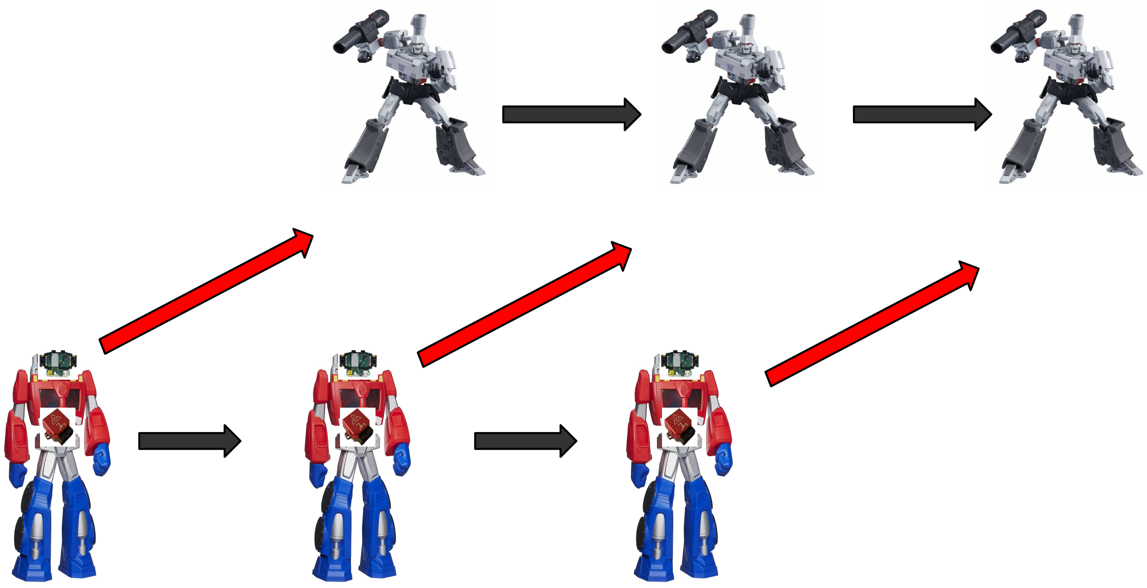


VINS Example



Target Tracking

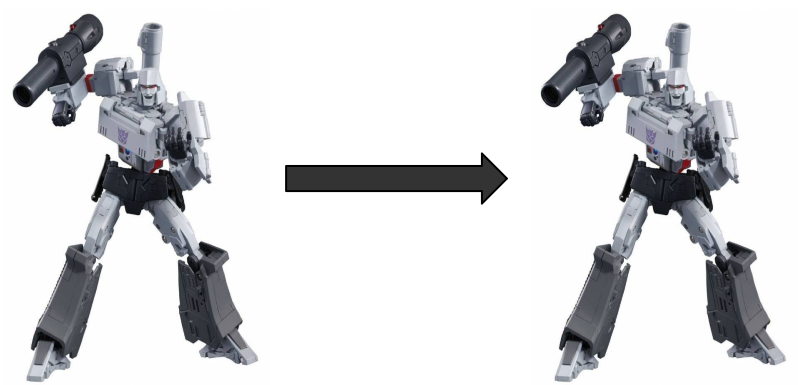
- In the standard VINS problem, we assume that the main goal is to simply navigate in a static, unknown environment
- What if our main goal is to instead track a moving target?
- "Optimus prime needs to be able to track the motion of Megatron in order to prevent him from doing evil"



Motion Model

- Unlike the static features, a target's state is changing due to motion
- Unlike the robot, we do not have access to the target's odometry measurements, and must instead rely on external measurements and a model for the target's motion
- Choice of motion model determines which parameters of the target need to be estimated

$$\mathbf{x}_{T_{k+1}} = \mathbf{f}\left(\mathbf{x}_{T_k}, \mathbf{n}_{T_k}\right)$$



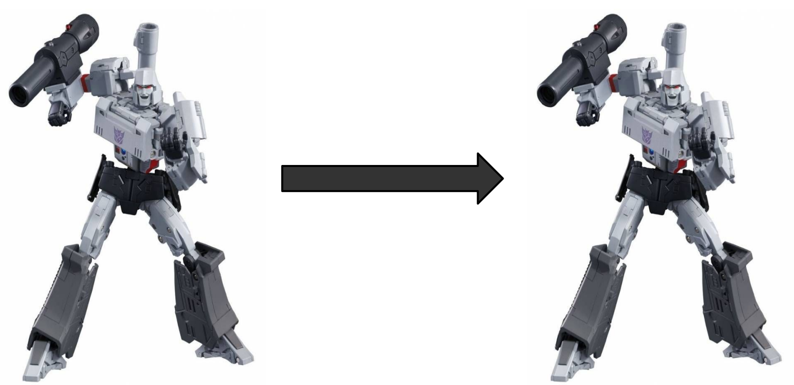
Constant Velocity Model

- Example model: constant velocity
- Requires that we estimate both the target's position and global velocity
- Choice of noise strength is used to capture a target's "predictability" based on the motion model

$$\mathbf{x}_T = \begin{bmatrix} {}^G\mathbf{p}_T \\ {}^G\mathbf{v}_T \end{bmatrix}$$

$$ {}^G\dot{\mathbf{p}}_T = {}^G\mathbf{v}_T $$

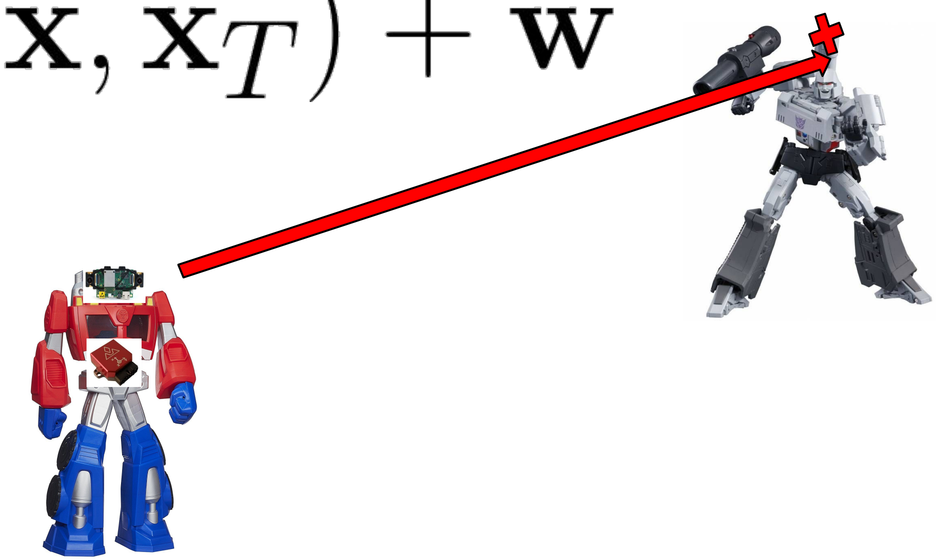
$$ {}^G\dot{\mathbf{v}}_T = \mathbf{n}_T $$



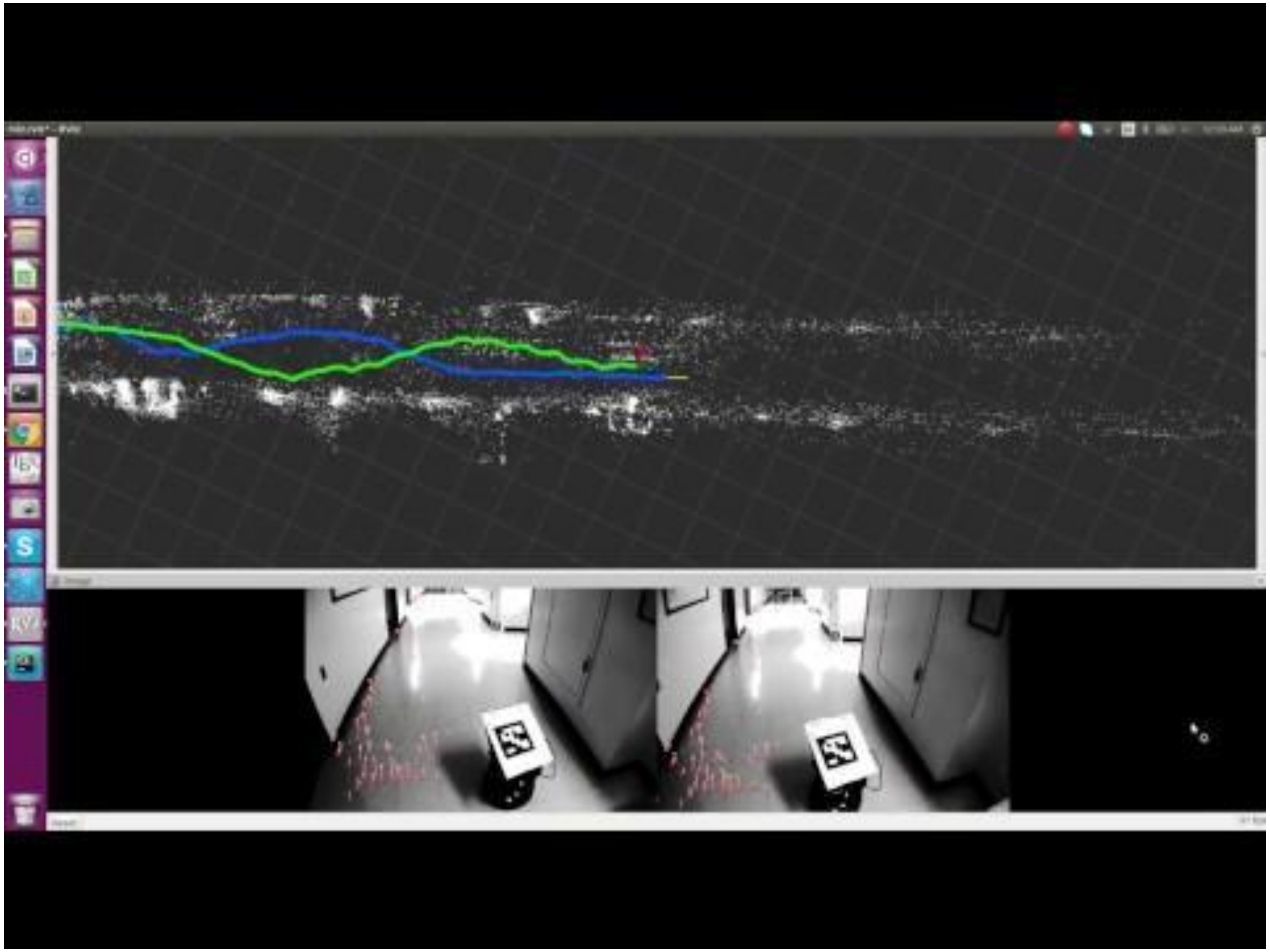
Target Measurements

- Measurement model treats the target as a point mass
- We assume that we can detect this reference point in our images
- Generates bearing measurements between the robot and target

$$\mathbf{z} = \mathbf{h}\left(\mathbf{x}, \mathbf{x}_T\right) + \mathbf{w}$$



Point Particle Target Tracking

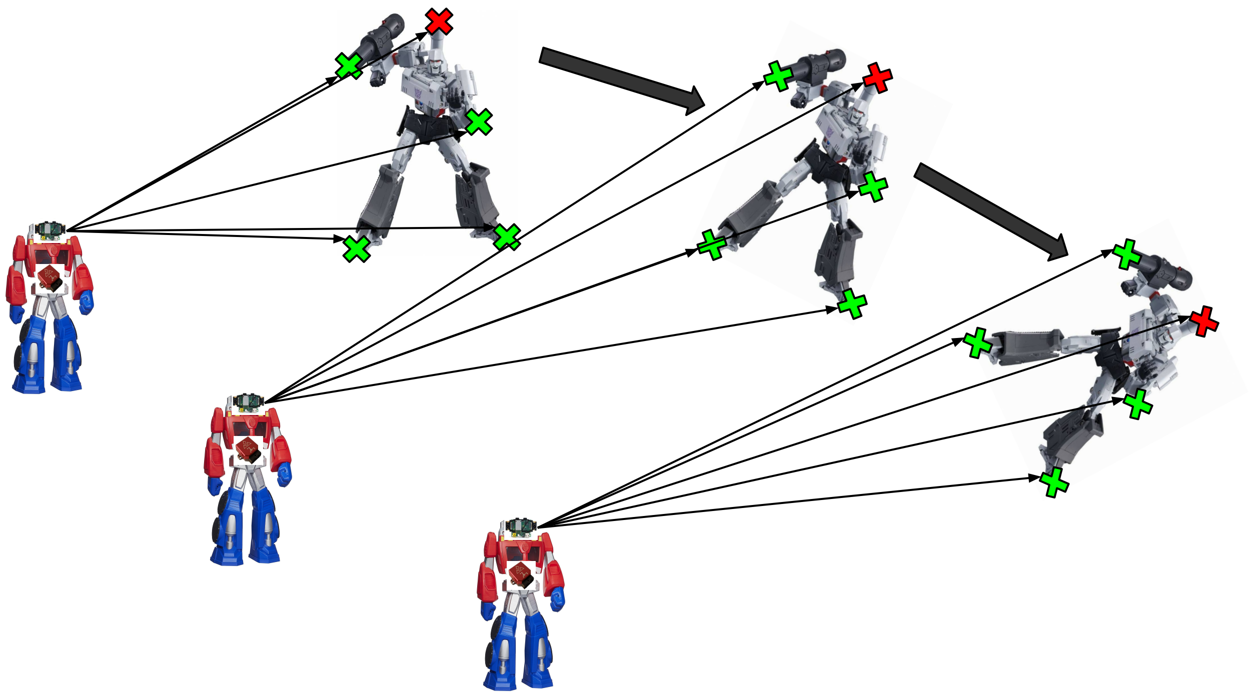


Cons of the Point Model

- Real targets rarely act as points. In addition, we often lose track of the reference point due changes of viewing angle

Rigid Body Target Tracking

- Solution: treat the target as a moving rigid body
- Position of features on the target wrt a reference point are static in the target's frame, and can be efficiently estimated

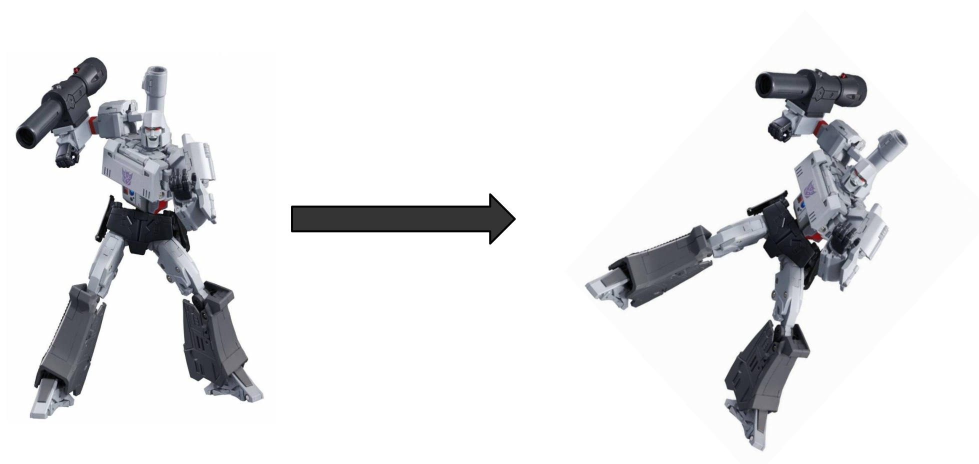


Constant Local Velocity Model

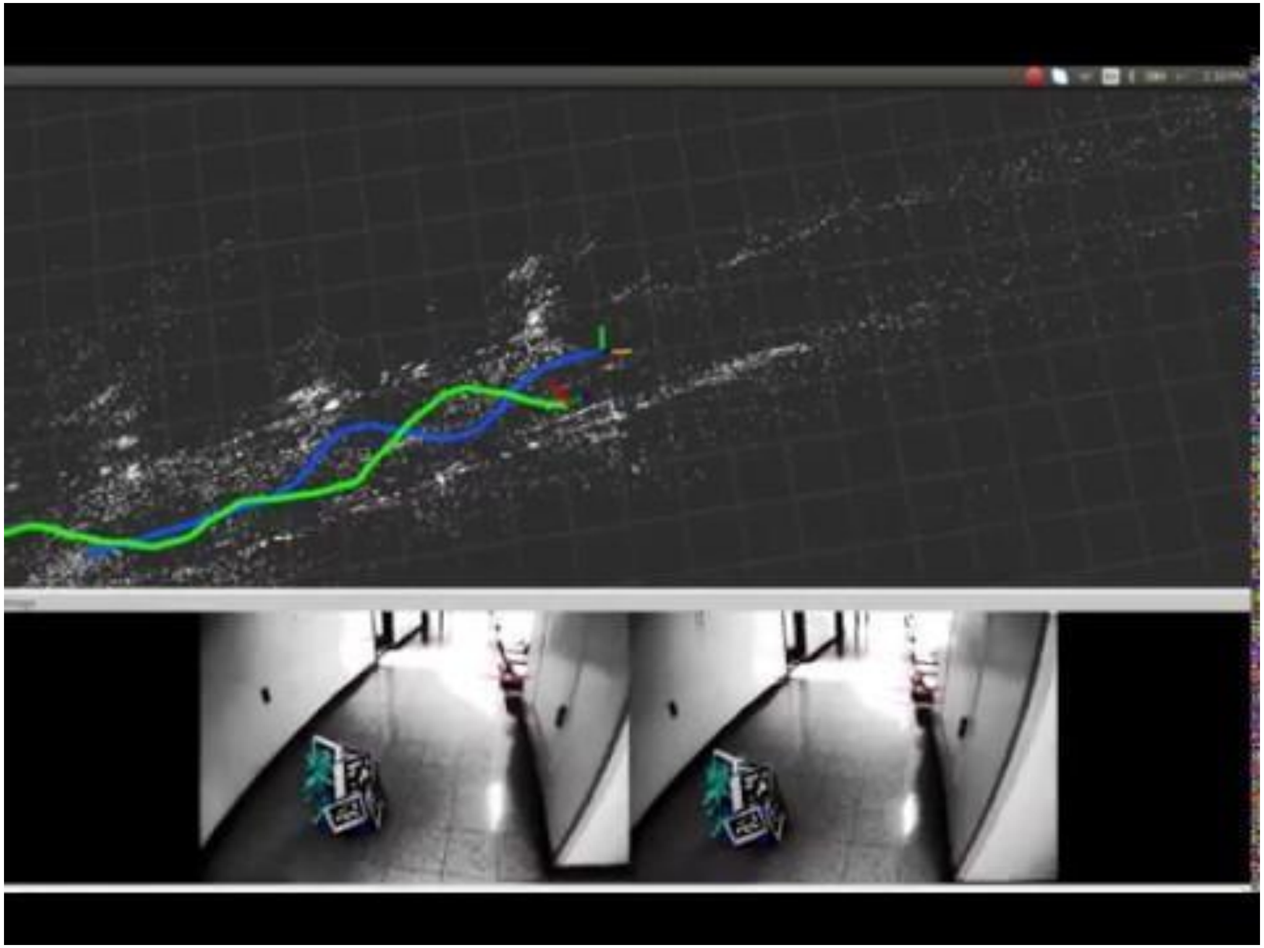
- With notion of pose, we can assign more "interesting" motion models
- Example: Constant local linear and angular velocity

$$\mathbf{x}_T = \begin{bmatrix} {}^T_G q \\ {}^G\mathbf{p}_T \\ {}^T\mathbf{v}_T \\ {}^T\mathbf{w} \end{bmatrix} \qquad {}^G\dot{\mathbf{p}}_T = {}^G_T\mathbf{R}\,{}^T\mathbf{v}_T \qquad {}^T_G\dot{\mathbf{R}} = -\lfloor {}^T\mathbf{w} \rfloor\, {}^T_G\mathbf{R}$$

$$ {}^T\dot{\mathbf{v}}_T = \mathbf{n}_T \qquad {}^T\dot{\mathbf{w}} = \mathbf{n}_{TW}$$



Rigid Body Tracking: Constant Local Velocity



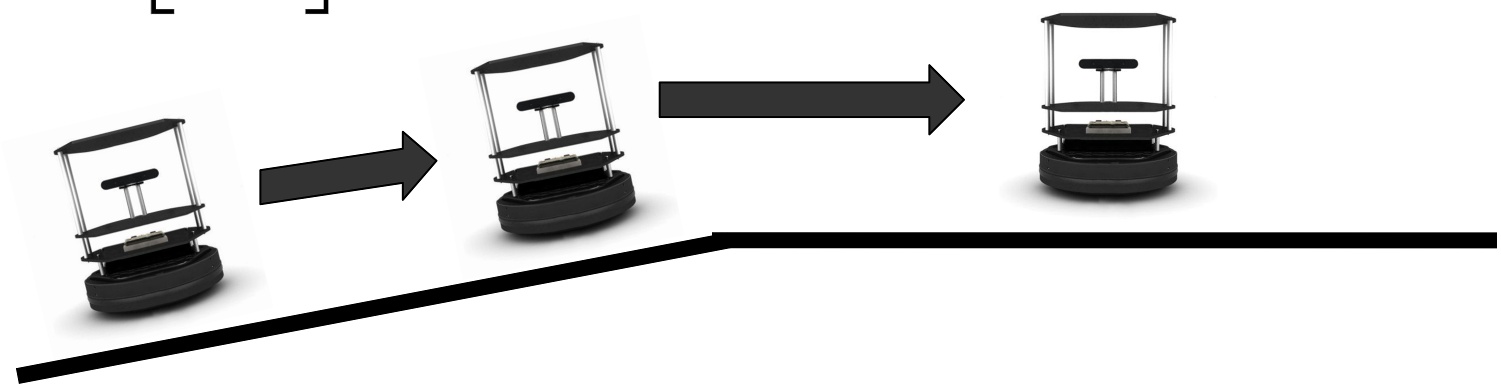
Pseudo-Unicycle Model

- Pseudo-Unicycle Model: vehicle moves “mostly” in its local x direction and with a rotation about its local z.
- Allows for the target to mostly evolve along planes

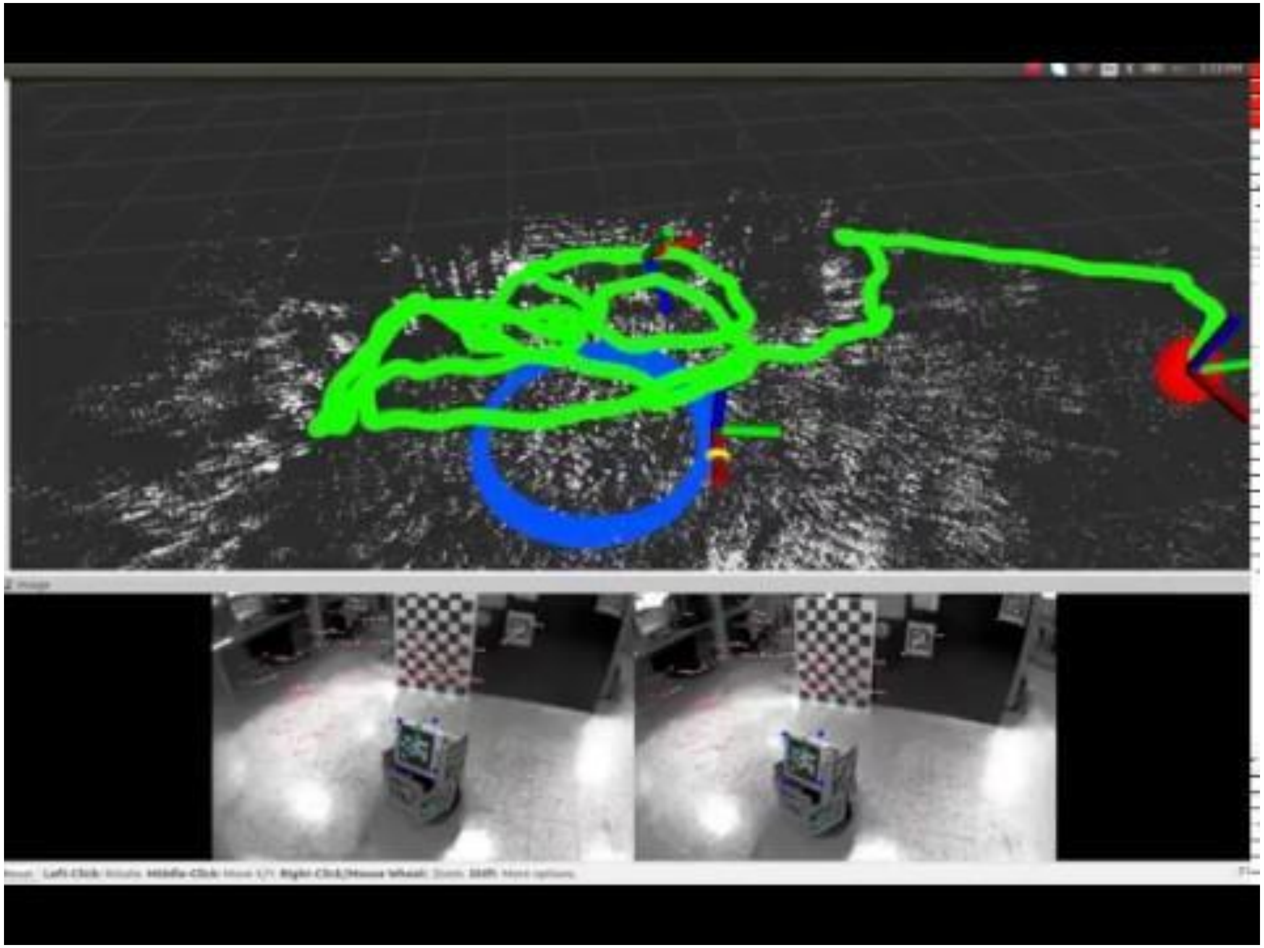
$$\mathbf{x}_T = \begin{bmatrix} {}^T_G q \\ {}^G\mathbf{p}_T \\ v_x \\ w_z \end{bmatrix}$$

$${}^G\dot{\mathbf{p}}_T = {}^G_T\mathbf{R} \begin{bmatrix} v_x \\ n_y \\ n_z \end{bmatrix} \qquad {}^T_G\dot{\mathbf{R}} = -\left\lfloor \begin{bmatrix} n_{wx} \\ n_{wy} \\ w_z \end{bmatrix} \right\rfloor {}^T_G\mathbf{R}$$

$$\dot{v}_x = n_x \qquad \dot{w}_z = n_{wz}$$



Rigid Body Tracking: Pseudo-Unicycle



Active Target Tracking