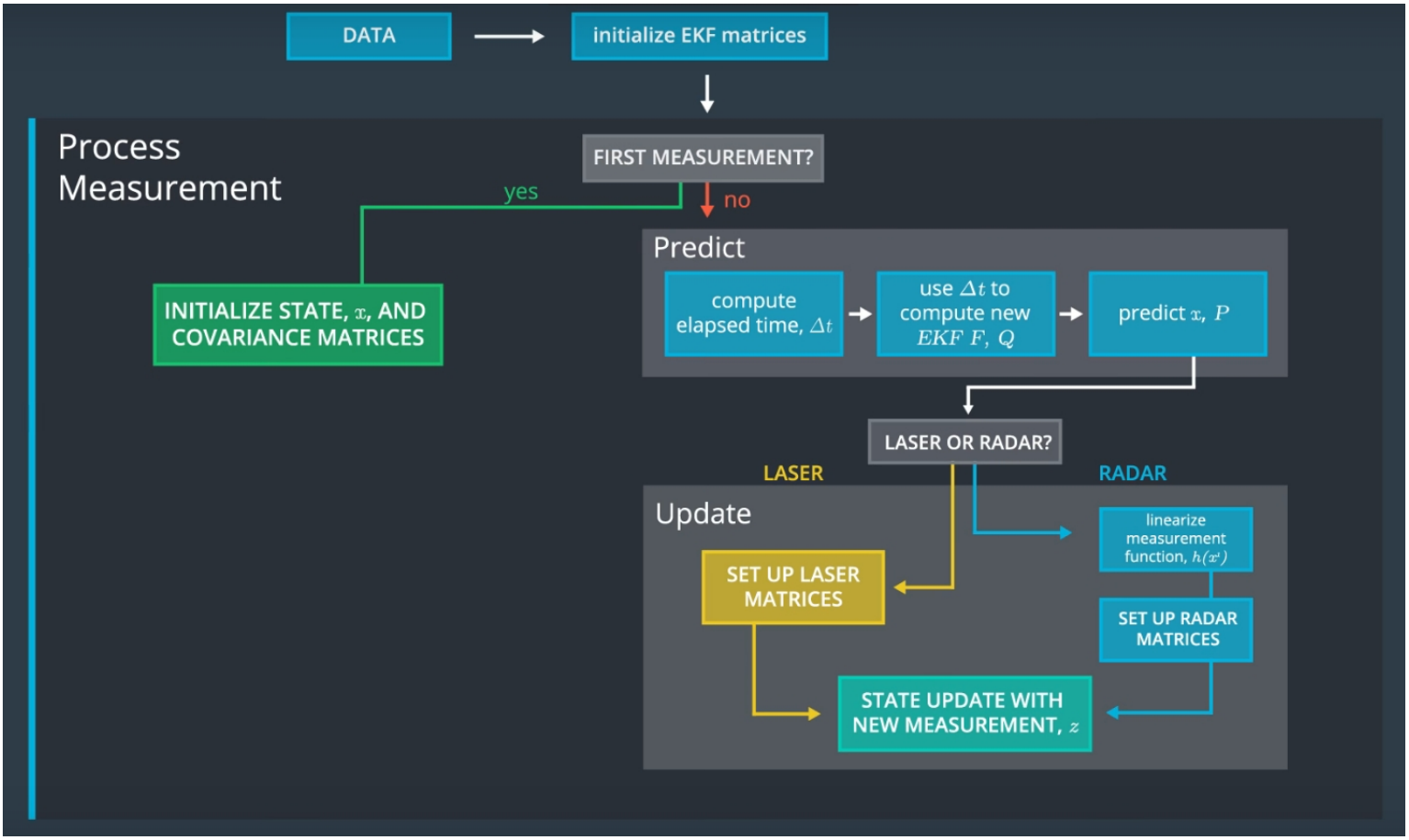
The goal in this project is to use noisy sensor data from RADAR and LASER (LIDAR) measurements for a moving object such as pedestrian or bicycle and track the object by using Extended Kalman Filter. There are pros and cons for each of them. For instance, LIDAR resolution is better than RADAR. However, RADAR works better in bad weather conditions. RADAR also can measure the velocity while LIDAR cannot.

Below image shows the measurement process (including first measurement, predict, and update with LASER or RADAR) which we learned from the lessons.



The structure of the coding is based on above flowchart. The main.cpp coordinates all the code. It communicates with simulator. It calls FusionEFK.cpp for initialization, update, and prediction. FusionEKF calls the functions for predict and update which is written in kalman_filter.cpp. Over there call for calculating RMSE (Root Mean Square Error) and Jacobian Matrix written in tools.cpp will happen.

I did the following coding for this project. The source for applying formulas and codes have been lessons (specially lesson 24) and cheat sheet pdf file for sensor fusion ekf reference.:

In FusuionEKF.cpp:

I did initialize the FuisnEKF by using learnings from section 11 in lesson 24:

```
H_laser_ << 1, 0, 0, 0,   //from section 11 section  in lesson 24
            0, 1, 0, 0;

Hj_ << 1, 1, 0, 0,
       1, 1, 0, 0,
       1, 1, 1, 1;
//the initial transition matrix F_
ekf_.F_ =  MatrixXd(4,4);// 4*4 Matrix (state transition)
ekf_.F_ << 1, 0, 1, 0,
           0, 1, 0, 1,
           0, 0, 1, 0,
           0, 0, 0, 1;

//state covariance matrix P
ekf_.P_ =  MatrixXd(4, 4);// 4*4 Matrix
ekf_.P_ << 1, 0, 0, 0,
           0, 1, 0, 0,
           0, 0, 1000, 0,
           0, 0, 0, 1000;
```

Then did initialize the state ekf_.x_ with the first measurement. And converted radar from polar to cartesian coordinates and initialize state:

```
  // first measurement
  cout << "EKF: " << endl;
  ekf_.x_ = VectorXd(4);
  ekf_.x_ << 1, 1, 1, 1;

  if (measurement_pack.sensor_type_ == MeasurementPackage::RADAR) {
    // TODO: Convert radar from polar to cartesian coordinates
    //         and initialize state.
     //ekf_.x_(0)=ro*cos(phi)
     //ekf_.x_(1)=ro*sin(phi)

    //Convert radar from polar to cartesian coordinates and initialize state.
    float ro     = measurement_pack.raw_measurements_(0);
    float phi    = measurement_pack.raw_measurements_(1);
    float ro_dot = measurement_pack.raw_measurements_(2);
    ekf_.x_(0) = ro     * cos(phi);
    ekf_.x_(1) = ro     * sin(phi);
    ekf_.x_(2) = ro_dot * cos(phi);
    ekf_.x_(3) = ro_dot * sin(phi);

  }
  else if (measurement_pack.sensor_type_ == MeasurementPackage::LASER) {
    // TODO: Initialize state.
    //ekf_.x_(0)=x
    //ekf_.x_(1)=y

    ekf_.x_(0) = measurement_pack.raw_measurements_(0);
    ekf_.x_(1) = measurement_pack.raw_measurements_(1);
```

Then I did coding for Prediction:

- I modified the F matrix by using learnings from section 9 in lesson 24
- I put noise_ax and noise_ay as 9 based on learnings from section 14 of lesson 24 (square 3)
- I set the process covariance matrix Q from learnings in section 10 of lesson 24

```
/*****************************************************************************
 * Prediction
 *****************************************************************************/

/**
 * TODO: Update the state transition matrix F according to the new elapsed time.
 * Time is measured in seconds.
 * TODO: Update the process noise covariance matrix.
 * Use noise_ax = 9 and noise_ay = 9 for your Q matrix.
 */

//compute the time elapsed between the current and previous measurements
float dt = (measurement_pack.timestamp_ - previous_timestamp_) / 1000000.0;	//dt - expressed in s
previous_timestamp_ = measurement_pack.timestamp_;

float dt_2 = dt   * dt;
float dt_3 = dt_2 * dt;
float dt_4 = dt_3 * dt;

//Modify the F matrix so that the time is integrated. section 9 of lesson 24
ekf_.F_(0, 2) = dt;
ekf_.F_(1, 3) = dt;

//set the acceleration noise components
float noise_ax = 9;     //In quiz in section 14 of lesson 24 is considered as 9 (square 3)
float noise_ay = 9;     //In quiz in section 14 of lesson 24 is considered as 9  (square 3)

//set the process covariance matrix Q.   section 10 of lesson 24
ekf_.Q_ = MatrixXd(4, 4);
ekf_.Q_ << dt_4 / 4 * noise_ax, 0, dt_3 / 2 * noise_ax, 0,
           0, dt_4 / 4 * noise_ay, 0, dt_3 / 2 * noise_ay,
           dt_3 / 2 * noise_ax, 0, dt_2*noise_ax, 0,
           0, dt_3 / 2 * noise_ay, 0, dt_2*noise_ay;

ekf_.Predict();  //go to kalman_filter.cpp  line 25
```

Then I did coding for Update. For Radar Update, used tools.cpp to calculate Jacobian. After return from tools.cpp, it puts H being the calculated Jacobian Hj of the extended kalman filter state vector:

```
  /*****************************************************************************
   * Update
   ****************************************************************************/

  /**
   * TODO:
   * - Use the sensor type to perform the update step.
   * - Update the state and covariance matrices.
   */

  if (measurement_pack.sensor_type_ == MeasurementPackage::RADAR) {
    // TODO: Radar updates
    Tools tools;  // uisng tools.cpp to calculate Jacobian
    Hj_ = tools.CalculateJacobian(ekf_.x_);
    ekf_.H_ = Hj_;     // the H we are using here ends up being the calculated Jacobian Hj of the ex
    ekf_.R_ = R_radar_;  // R is just the
    R radar matrix we defined above
    ekf_.UpdateEKF(measurement_pack.raw_measurements_);   //go to kalman_filter.cpp  line 36

  } else {
    // TODO: Laser updates
    ekf_.H_ = H_laser_;
    ekf_.R_ = R_laser_;
    ekf_.Update(measurement_pack.raw_measurements_);
  }

  // print the output
  cout << "x_ = " << ekf_.x_ << endl;
  cout << "P_ = " << ekf_.P_ << endl;
}
```

In Kalman_filter.cpp:

- Did coding for predict the state by using learnings from sections 9 and 10 from lesson 24.
- Did coding for updating the state by using Kalman Filter equations, from learnings in sections 8 and 15 of lesson 24

```
   * TODO: predict the state
   */

 x_ = F_ * x_;                          //section 9 in lesson 24
  MatrixXd Ft = F_.transpose();         // section 10 in lesson 24
  P_ = F_ * P_ * Ft + Q_;

}

void KalmanFilter::Update(const VectorXd &z) {
  /**
   * TODO: update the state by using Kalman Filter equations
   */

  VectorXd z_pred = H_ * x_;                            // section 8 of leson 24 line 83. H*x gives us our z pre
  VectorXd y = z - z_pred;
  MatrixXd Ht = H_.transpose();
  MatrixXd S = H_ * P_ * Ht + R_;
  MatrixXd Si = S.inverse();
  MatrixXd PHt = P_ * Ht;
  MatrixXd K = PHt * Si;

  //new state
  x_ = x_ + (K * y);
  long x_size = x_.size();
  MatrixXd I = MatrixXd::Identity(x_size, x_size);
  P_ = (I - K * H_) * P_;

}
```

```
void KalmanFilter::UpdateEKF(const VectorXd &z) {
  /**
   * TODO: update the state by using Extended Kalman Filter equations
   */

  // section 15 of lesson 24
  float rho = sqrt(x_(0)*x_(0) + x_(1)*x_(1));  // x values generated by prediction step above
  float phi = atan2(x_(1), x_(0));
  float rho_dot;
  if (fabs(rho) < 0.0001) {
    rho_dot = 0;
  } else {
    rho_dot = (x_(0)*x_(2) + x_(1)*x_(3))/rho;
  }
  VectorXd z_pred(3);
  z_pred << rho, phi, rho_dot;
  VectorXd y = z - z_pred;   // last time above y was z-h*x

  // in section 8 of lesson 24
  MatrixXd Ht = H_.transpose();
  MatrixXd S = H_ * P_ * Ht + R_;
  MatrixXd Si = S.inverse();
  MatrixXd PHt = P_ * Ht;
  MatrixXd K = PHt * Si;

   // new state
   x_ = x_ + (K * y);
  long x_size = x_.size();
  MatrixXd I = MatrixXd::Identity(x_size, x_size);
  P_ = (I - K * H_) * P_;
```

In tools.cpp:

- Calculatged the RMSE based on learnings from section 24 of lesson 24:

```
  * TODO: Calculate the RMSE here.
  */


VectorXd rmse(4);                              // section 24 of lesson 24
rmse << 0,0,0,0;

// check the validity of the following inputs:
//  * the estimation vector size should not be zero
//  * the estimation vector size should equal ground truth vector size
if (estimations.size() != ground_truth.size()
    || estimations.size() == 0) {
  std::cout << "Invalid estimation or ground_truth data" << std::endl;
  return rmse;
}

// accumulate squared residuals
for (unsigned int i=0; i < estimations.size(); ++i) {

  VectorXd residual = estimations[i] - ground_truth[i];

  // coefficient-wise multiplication
  residual = residual.array()*residual.array();
  rmse += residual;
}

// calculate the mean
rmse = rmse/estimations.size();

// calculate the squared root
rmse = rmse.array().sqrt();

// return the result
return rmse;
```

- Calculated jacobian:

```
 * Calculate a Jacobian here.
 */



MatrixXd Hj(3,4);

Hj << 0,0,0,0,
      0,0,0,0,
      0,0,0,0;

     // section 20 of lesson 24
// recover state parameters
float px = x_state(0);
float py = x_state(1);
float vx = x_state(2);
float vy = x_state(3);

// pre-compute a set of terms to avoid repeated calculation
float c1 = px*px+py*py;
float c2 = sqrt(c1);
float c3 = (c1*c2);

// check division by zero
if (fabs(c1) < 0.0001) {
  std::cout << "CalculateJacobian () - Error - Division by Zero" << std::endl;
  return Hj;
}

// compute the Jacobian matrix
Hj << (px/c2), (py/c2), 0, 0,
    -(py/c1), (px/c1), 0, 0,
    py*(vx*py - vy*px)/c3, px*(px*vy - py*vx)/c3, px/c2, py/c2;

return Hj;
```

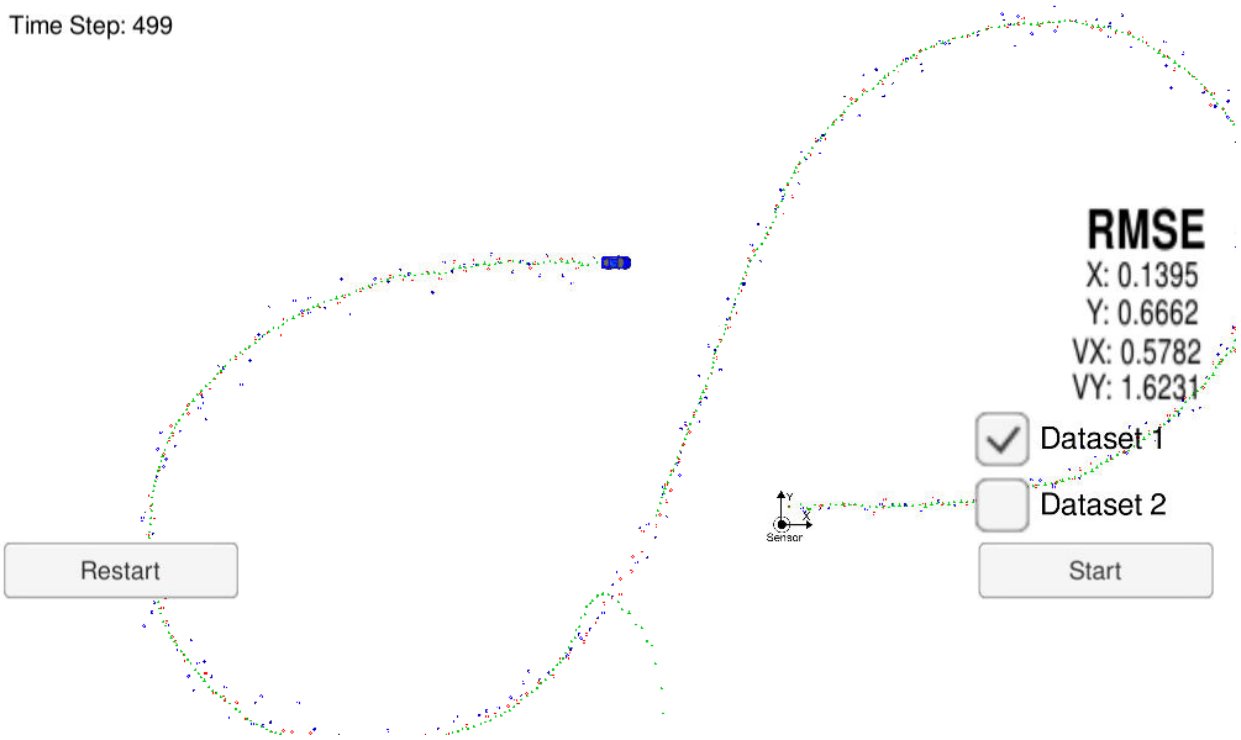
Then I used the following commands in terminal to compile and build the main program(from the project top directory):

- mkdir build && cd build
- cmake .. && make
- ./ExtendedKF

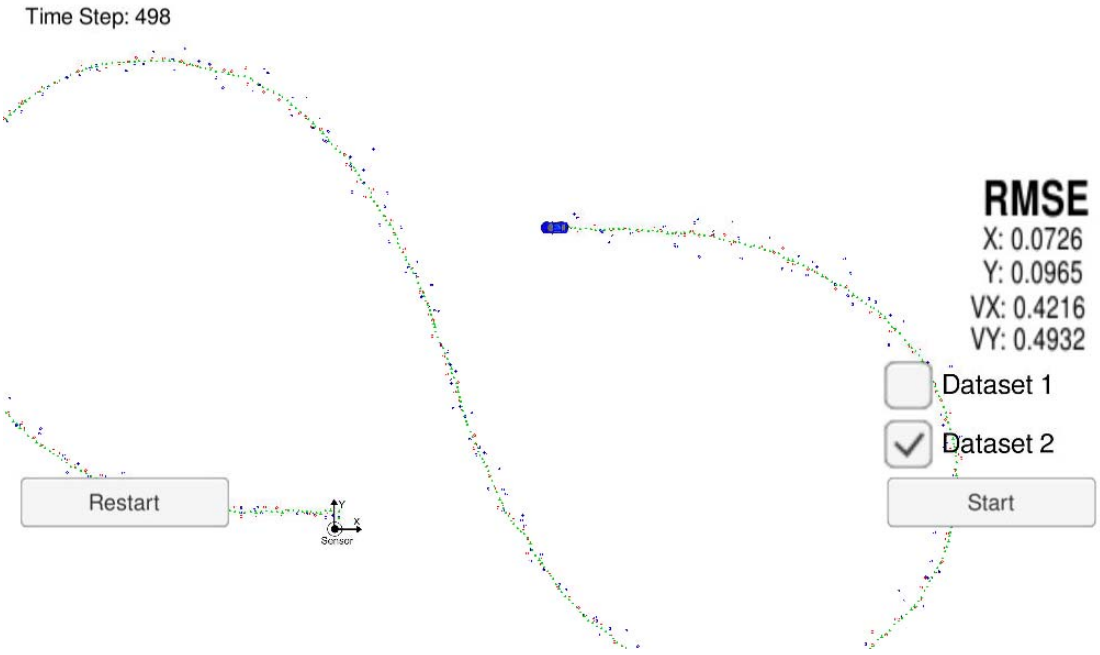
Simulation results:

Below are results of simulation for two data sets:

- Data set1:



- Data set2:



As seen above, results for dataset2 have better RMSE and are less than minimum required by proejct rubric (RMSE <= [px .11, py .11, vx 0.52, vy 0.52])

However for dataset1, the results are above the minimum requiremets. I don't have an idea of how t ofix this issue and would ask the instructors and reviewrs to provide me with an idea

Overal, this was a great experience and I liked to see the result of my codes in simulator.

Self-Driving Car ND - Sensor Fusion - Extended Kalman Filters

Udacity and Mercedes

February 27, 2017

1 Introduction

No equations.

2 Lesson Map

No equations.

3 Estimation Problem Refresh

No equations.

4 Measurement Update Quiz

No equations.

5 Kalman Filter Equations In C++

The state transition function is

$$x' = f(x) + \nu = Fx + \underbrace{Bu}_{=0} + \nu \tag{1}$$

The motion control, Bu, is zero in our case, so our state transition function is simplified to:

$$x' = Fx + \nu \tag{2}$$

The measurement function is

$$z = h(x') + \omega = Hx' + \omega \tag{3}$$

where,
ν is the process noise. It is a Gaussian with zero mean and covariance matrix Q:

$$\nu \sim N(0, Q) \tag{4}$$

ω is the measurement noise. It is a Gaussian with zero mean and covariance matrix R:

$$\omega \sim N(0, R) \tag{5}$$

The state vector:

$$x = \begin{pmatrix} p \\ v \end{pmatrix} \tag{6}$$

Linear motion:

$$p' = p + v\Delta t \tag{7}$$

$$v' = v \tag{8}$$

The state prediction function:

$$\begin{pmatrix} p' \\ v' \end{pmatrix} = \begin{pmatrix} 1 & \Delta t \\ 0 & 1 \end{pmatrix} \begin{pmatrix} p \\ v \end{pmatrix} \tag{9}$$

The measurement function:

$$z = \begin{pmatrix} 1 & 0 \end{pmatrix} \begin{pmatrix} p' \\ v' \end{pmatrix} \tag{10}$$

5.1 Kalman Filter Algorithm

Prediction:

$$x' = Fx + u \tag{11}$$

$$P' = FPF^T + Q \tag{12}$$

Measurement Update:

$$y = z - Hx' \tag{13}$$

$$S = HP'H^T + R \tag{14}$$

$$K = P'H^T S^{-1} \tag{15}$$

$$x = x' + Ky \tag{16}$$

$$P = (I - KH)P' \tag{17}$$

6 Kalman Filter Equation in C++ Programming Assignment

$$x = \begin{pmatrix} p_x \\ p_y \\ v_x \\ v_y \end{pmatrix} \tag{18}$$

7 Measurement Update Quiz

No equations.

8 State Prediction - 4D Case

No equations.

9 Uncertainty Depending on Time Delay Quiz

No equations.

10 Uncertainty Depending on Acceleration Quiz

No equations.

11 Process Covariance Matrix

Linear Motion Model - 2D:

$$x' = Fx + \nu \tag{19}$$

$$\begin{cases} p'_x = p_x + v_x \Delta t + \nu_{px} \\ p'_y = p_y + v_y \Delta t + \nu_{py} \\ v'_x = v_x + \nu_{vx} \\ v'_y = v_y + \nu_{vy} \end{cases} \tag{20}$$

Can be expressed in a matrix form as:

$$\begin{pmatrix} p'_x \\ p'_y \\ v'_x \\ v'_y \end{pmatrix} = \begin{pmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} p_x \\ p_y \\ v_x \\ v_y \end{pmatrix} + \begin{pmatrix} \nu_{px} \\ \nu_{py} \\ \nu_{vx} \\ \nu_{vy} \end{pmatrix} \tag{21}$$

11.1 Deriving the 2D velocity with a constant acceleration

From the kinematic formulas we can define the acceleration as the change in the velocity over the elapsed time, so the current velocity minus previous velocity over the elapsed time.

$$a = \frac{\Delta v}{\Delta t} = \frac{v_{k+1} - v_k}{\Delta t} \tag{22}$$

Then we can derive the current velocity as the previous velocity plus the acceleration a times Δt.

$$v_{k+1} = v_k + a\Delta t \tag{23}$$

Generalized in 2D we have both speed components described as follows:

$$\begin{cases} v'_x = v_x + a_x \Delta t \\ v'_y = v_y + a_y \Delta t \end{cases} \tag{24}$$

11.2 Deriving the relation between the previous location and current position in 2D when having a motion with constant acceleration

Similarly we know that the displacement Δp equals the average velocity v_{avg} times Δt.

$$\Delta p = p_{k+1} - p_k = v_{avg} \Delta t \tag{25}$$

which is the previous velocity v_{k+1} plus current velocity v_k over two.

$$p_{k+1} - p_k = \frac{v_{k+1} + v_k}{2} \Delta t \tag{26}$$

$$p_{k+1} = p_k + \frac{v_{k+1} + v_k}{2} \Delta t \tag{27}$$

And as we already saw, the current velocity v_{k+1} is our old velocity v_k plus the acceleration a times Δt.

$$v_{k+1} = v_k + a\Delta t \tag{28}$$

so we can replace the v_{k+1} with $v_k + a\Delta t$ in previous equations:

$$p_{k+1} = p_k + \frac{v_k + a\Delta t + v_k}{2} \Delta t \tag{29}$$

$$p_{k+1} = p_k + \frac{2v_k \Delta t}{2} + \frac{a\Delta t^2}{2} \tag{30}$$

And finally we get the full relation between the previous location p_k and current location p_{k+1}:

$$p_{k+1} = p_k + v_k \Delta t + \frac{a\Delta t^2}{2} \tag{31}$$

Going in 2D, we'll have both horizontal and vertical directions:

$$\begin{cases} p'_x = p_x + v_x \Delta t + \frac{a_x \Delta t^2}{2} \\ p'_y = p_y + v_y \Delta t + \frac{a_y \Delta t^2}{2} \end{cases} \tag{32}$$

11.3 The 2D motion model with constant acceleration

Combining both 2D position and 2D velocity equations previously deducted formulas we have:

$$\begin{cases} p'_x = p_x + v_x\Delta t + \frac{a_x\Delta t^2}{2} \\ p'_y = p_y + v_y\Delta t + \frac{a_y\Delta t^2}{2} \\ v'_x = v_x + a_x\Delta t \\ v'_y = v_y + a_y\Delta t \end{cases} \tag{33}$$

11.4 Process Noise in 2D

Since the acceleration is unknown we can add it to the noise component, and this random noise would be expressed analytically as the last terms in the equation derived above. So, we have a random acceleration vector ν in this form:

$$\nu = \begin{pmatrix} \nu_{px} \\ \nu_{py} \\ \nu_{vx} \\ \nu_{vy} \end{pmatrix} = \begin{pmatrix} \frac{a_x\Delta t^2}{2} \\ \frac{a_y\Delta t^2}{2} \\ a_x\Delta t \\ a_y\Delta t \end{pmatrix} \tag{34}$$

which is described by a zero mean and a covariance matrix Q.

$$\nu \sim N(0, Q) \tag{35}$$

The vector ν can be decomposed into two components a 4 by 2 matrix G which does not contain random variables and a 2 by 1 matrix a which contains the random acceleration components:

$$\nu = \begin{pmatrix} \frac{a_x\Delta t^2}{2} \\ \frac{a_y\Delta t^2}{2} \\ a_x\Delta t \\ a_y\Delta t \end{pmatrix} = \underbrace{\begin{pmatrix} \frac{\Delta t^2}{2} & 0 \\ 0 & \frac{\Delta t^2}{2} \\ \Delta t & 0 \\ 0 & \Delta t \end{pmatrix}}_{G} \underbrace{\begin{pmatrix} a_x \\ a_y \end{pmatrix}}_{a} = Ga \tag{36}$$

Δt is computed at each Kalman Filter step and the acceleration is a random vector with zero mean and standard deviations σ_{ax} and σ_{ay}.

Based on our noise vector we can define now the new covariance matrix Q. The covariance matrix is defined as the expectation value of the noise vector ν times the noise vector ν transpose. So let's write this down:

$$Q = E[\nu\nu^T] = E[Gaa^TG^T] \tag{37}$$

As G does not contain random variables, we can put it outside the expectation calculation.

$$Q = GE[aa^T]G^T = G\begin{pmatrix} \sigma_{ax}^2 & \sigma_{axy} \\ \sigma_{axy} & \sigma_{ay}^2 \end{pmatrix} G^T = GQ_\nu G^T \tag{38}$$

This leaves us with three statistical moments:

- the expectation of ax times ax, which is the variance of ax: σ_{ax}^2.
- the expectation of ay times ay, which is the variance of ay: σ_{ay}^2 squared.
- And the expectation of ax times ay, which is the covariance of ax and ay: σ_{axy}.

a_x and a_y are assumed uncorrelated noise processes. This means that the covariance σ_{axy} in Q_ν is zero:

$$Q_\nu = \begin{pmatrix} \sigma_{ax}^2 & \sigma_{axy} \\ \sigma_{axy} & \sigma_{ay}^2 \end{pmatrix} = \begin{pmatrix} \sigma_{ax}^2 & 0 \\ 0 & \sigma_{ay}^2 \end{pmatrix} \tag{39}$$

So after combining everything in one matrix we obtain our 4 by 4 Q matrix:

$$Q = GQ_\nu G^T = \begin{pmatrix} \frac{\Delta t^4}{4}\sigma_{ax}^2 & 0 & \frac{\Delta t^3}{2}\sigma_{ax}^2 & 0 \\ 0 & \frac{\Delta t^4}{4}\sigma_{ay}^2 & 0 & \frac{\Delta t^3}{2}\sigma_{ay}^2 \\ \frac{\Delta t^3}{2}\sigma_{ax}^2 & 0 & \Delta t^2\sigma_{ax}^2 & 0 \\ 0 & \frac{\Delta t^3}{2}\sigma_{ay}^2 & 0 & \Delta t^2\sigma_{ay}^2 \end{pmatrix} \tag{40}$$

11.5 other equations

Note: Some authors describe Q as the complete process noise covariance matrix. And some authors describe Q as the covariance matrix of the individual noise processes. In our case, the covariance matrix of the individual noise processes matrix is called Q_ν. So we should be aware of that.

12 Process Covariance Matrix Text

No equations.

13 Laser Measurements

No equations.

14 Find Out the H Matrix

14.1 Quiz question

$$\begin{pmatrix} p_x \\ p_y \end{pmatrix} = (?) \begin{pmatrix} p'_x \\ p'_y \\ v'_x \\ v'_y \end{pmatrix} \tag{41}$$

14.2 Quiz answer

$$\begin{pmatrix} p_x \\ p_y \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} p'_x \\ p'_y \\ v'_x \\ v'_y \end{pmatrix} \tag{42}$$

15 Find out the Dimension of R

15.1 Quiz question

Now, can you determine what is the dimensionality of the measurement noise covariance matrix R.

15.2 Quiz answer

$$R = E[\omega\omega^T] = \begin{pmatrix} \sigma_{px}^2 & 0 \\ 0 & \sigma_{py}^2 \end{pmatrix} \tag{43}$$

16 Programming Assignment

$$R = \begin{pmatrix} 0.2 & 0 \\ 0 & 0.2 \end{pmatrix} \tag{44}$$

$$R = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix} \tag{45}$$

16.1 Other Equations

$$a_x \sim N(0, \sigma_{ax}^2) \tag{46}$$

$$a_y \sim N(0, \sigma_{ay}^2) \tag{47}$$

17 Radar Measurements

The state transition function:

$$x' = f(x) + \nu \tag{48}$$

The measurement function:

$$z = h(x') + \omega \tag{49}$$

The state transition function will be exactly what you've designed in the lidar case.

However our new Radar sees the world differently. Instead of a 2D pose (p_x, p_y), the radar can directly measure the object range ρ, bearing φ and range rate $\dot{\rho}$:

$$z = \begin{pmatrix} \rho \\ \varphi \\ \dot{\rho} \end{pmatrix} \tag{50}$$

- The range ρ is the radial distance from the origin to our pedestrian. So we can always define a ray which extends from the origin to our object position.
- The bearing φ is the angle between the ray and x direction.
- And the range rate $\dot{\rho}$, also known as Doppler or radial velocity is the velocity along this ray.

And similar to our motion model, the radar observations are corrupted by a zero-mean random noise $\omega \sim N(0, R)$. Considering that the three measurement components of the measurement vector z are not cross-correlated, the radar measurement covariance matrix R becomes a 3 by 3 diagonal matrix:

$$R = \begin{pmatrix} \sigma_\rho^2 & 0 & 0 \\ 0 & \sigma_\varphi^2 & 0 \\ 0 & 0 & \sigma_{\dot{\rho}}^2 \end{pmatrix} \tag{51}$$

And the next question is what is the measurement function $h(x')$ that maps the predicted state x' into the measurement space?

$$\begin{pmatrix} \rho \\ \varphi \\ \dot{\rho} \end{pmatrix} \xleftarrow{h(x')} \begin{pmatrix} p'_x \\ p'_y \\ v'_x \\ v'_y \end{pmatrix} \tag{52}$$

And that's how I defined the h function which basically specifies how the predicted position and speed can be related to the observed range ρ, bearing φ and range rate $\dot{\rho}$:

$$h(x') = \begin{pmatrix} \sqrt{p'^2_x + p'^2_y} \\ \arctan(p'_y / p'_x) \\ \frac{p'_x v'_x + p'_y v'_y}{\sqrt{p'^2_x + p'^2_y}} \end{pmatrix} \tag{53}$$

17.1 Deriving the Radar Measurement Function

The measurement function is composed of three components which show how the predicted state $x' = (p'_x, p'_y, v'_x, v'_y)^T$ is mapped into the measurement space $z = (\rho, \varphi, \dot{\rho})^T$:

1. The range ρ is the distance to the pedestrian which can be defined as:

$$\rho = \sqrt{p_x^2 + p_y^2} \tag{54}$$

2. φ is the angle between ρ and x direction and can be defined as:

$$\varphi = \arctan(p_y / p_x) \tag{55}$$

3. Range rate $\dot{\rho(t)}$ derivation:

- Method 1: Generally we can explicitly describe the range ρ as a function of time:

$$\rho(t) = \sqrt{p_x(t)^2 + p_y(t)^2} \tag{56}$$

The range rate $\dot{\rho(t)}$ is defined as time rate of change of the range ρ, and it can be described as the time derivative of ρ:

$$\dot{\rho} = \frac{\partial \rho(t)}{\partial t} = \frac{\partial}{\partial t}\sqrt{p_x(t)^2 + p_y(t)^2} = \frac{1}{2\sqrt{p_x(t)^2 + p_y(t)^2}}(\frac{\partial}{\partial t}p_x(t)^2 + \frac{\partial}{\partial t}p_y(t)^2) = \frac{1}{2\sqrt{p_x(t)^2 + p_y(t)^2}}(2p_x(t)\frac{\partial}{\partial t}p_x(t) + 2p_y(t)\frac{\partial}{\partial t}p_y(t))$$

$\frac{\partial}{\partial t}p_x(t)$ is nothing else than $v_x(t)$, similarly $\frac{\partial}{\partial t}p_y(t)$ is $v_y(t)$. So we have:

$$\dot{\rho} = \frac{\partial \rho(t)}{\partial t} = \frac{1}{2\sqrt{p_x(t)^2 + p_y(t)^2}}(2p_x(t)v_x(t)+2p_y(t)v_y(t)) = \frac{2(p_x(t)v_x(t) + p_y(t)v_y(t))}{2\sqrt{p_x(t)^2 + p_y(t)^2}} = \frac{p_x(t)v_x(t) + p_y(t)v_y(t)}{\sqrt{p_x(t)^2 + p_y(t)^2}}$$

For the simplicity we just use the following notation:

$$\dot{\rho} = \frac{p_x v_x + p_y v_y}{\sqrt{p_x^2 + p_y^2}} \tag{57}$$

- Method 2: The range rate $\dot{\rho}$ can be seen as a scalar projection of the velocity vector $\vec{v}$ onto $\vec{\rho}$. Both $\vec{\rho}$ and $\vec{v}$ are 2D vectors defined as:

$$\vec{\rho} = \begin{pmatrix} p_x \\ p_y \end{pmatrix}, \vec{v} = \begin{pmatrix} v_x \\ v_y \end{pmatrix} \tag{58}$$

The scalar projection of the velocity vector $\vec{v}$ onto $\vec{\rho}$ is defined as:

$$\dot{\rho} = \frac{\vec{v}\vec{\rho}}{|\vec{\rho}|} = \frac{\begin{pmatrix} v_x \\ v_y \end{pmatrix}\begin{pmatrix} p_x \\ p_y \end{pmatrix}}{\sqrt{p_x^2 + p_y^2}} = \frac{p_x v_x + p_y v_y}{\sqrt{p_x^2 + p_y^2}} \tag{59}$$

where $|\vec{\rho}|$ is the length of $\vec{\rho}$. In our case it is actually the range, so $\rho = |\vec{\rho}|$

18 Mapping with a Nonlinear Function Quiz

No equations.

19 Extended Kalman Filter

No equations.

20 Linearization Example

20.0.1 Linearization of a function

The Extended Kalman Filter uses a method called first order Taylor expansion

$$h(x) \approx h(\mu) + \frac{\partial h(\mu)}{\partial x}(x - \mu) \tag{60}$$

20.1 20_quiz_linearization_example

$$h(x) = \arctan(x) \tag{61}$$

$$\frac{\partial h(x)}{\partial x} = \frac{\partial \arctan(x)}{\partial x} = \frac{1}{1+x^2} \tag{62}$$

20.1.1 quiz answer

$$h(x) = \arctan(x) \tag{63}$$

$$h(x) \approx h(\mu) + \frac{\partial h(\mu)}{\partial x}(x-\mu) = \arctan(\mu) + \frac{1}{1+\mu^2}(x-\mu) \tag{64}$$

In our example $\mu = 0$, therefore:

$$h(x) \approx \arctan(0) + \frac{1}{1+0}(x-0) = x \tag{65}$$

So, the function $h(x) = \arctan(x)$ will be approximated by a line:

$$h(x) \approx x \tag{66}$$

21 Jacobian Matrix

State transition function:

$$x' = f(x) + \nu \tag{67}$$

Measurement function:

$$z = h(x') + \omega \tag{68}$$

where we consider that $f(x)$ and $h(x)$ are nonlinear and can be linearized as:

$$f(x) \approx f(\mu) + \underbrace{\frac{\partial f(\mu)}{\partial x}}_{F_j}(x-\mu) \tag{69}$$

$$h(x) \approx h(\mu) + \underbrace{\frac{\partial h(\mu)}{\partial x}}_{H_j}(x-\mu) \tag{70}$$

The derivative of $f(x)$ and $h(x)$ with respect to x are called Jacobians:

$$\frac{\partial f(\mu)}{\partial x} = F_j \tag{71}$$

$$\frac{\partial h(\mu)}{\partial x} = H_j \tag{72}$$

and it is going to be a matrix containing all the partial derivatives:

$$H_j = \begin{bmatrix} \frac{\partial h_1}{\partial x_1} & \frac{\partial h_1}{\partial x_2} & \cdots & \frac{\partial h_1}{\partial x_n} \\ \frac{\partial h_2}{\partial x_1} & \frac{\partial h_2}{\partial x_2} & \cdots & \frac{\partial h_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial h_m}{\partial x_1} & \frac{\partial h_m}{\partial x_2} & \cdots & \frac{\partial h_m}{\partial x_n} \end{bmatrix} \tag{73}$$

To be more specific, I know that my function h describes three components: range ρ, bearing φ and range-rate $\dot{\rho}$, and my state is a vector with four components p_x, p_y, v_x, v_y. Then the Jacobian matrix H_j is going to be a matrix with 3 rows and four columns:

$$H_j = \begin{bmatrix} \frac{\partial \rho}{\partial p_x} & \frac{\partial \rho}{\partial p_y} & \frac{\partial \rho}{\partial v_x} & \frac{\partial \rho}{\partial v_y} \\ \frac{\partial \varphi}{\partial p_x} & \frac{\partial \varphi}{\partial p_y} & \frac{\partial \varphi}{\partial v_x} & \frac{\partial \varphi}{\partial v_y} \\ \frac{\partial \dot{\rho}}{\partial p_x} & \frac{\partial \dot{\rho}}{\partial p_y} & \frac{\partial \dot{\rho}}{\partial v_x} & \frac{\partial \dot{\rho}}{\partial v_y} \end{bmatrix} \tag{74}$$

After computing all these partial derivatives we have:

$$H_j = \begin{bmatrix} \frac{p_x}{\sqrt{p_x^2+p_y^2}} & \frac{p_y}{\sqrt{p_x^2+p_y^2}} & 0 & 0 \\ -\frac{p_y}{p_x^2+p_y^2} & \frac{p_x}{p_x^2+p_y^2} & 0 & 0 \\ \frac{p_y(v_x p_y - v_y p_x)}{(p_x^2+p_y^2)^{3/2}} & \frac{p_x(v_y p_x - v_x p_y)}{(p_x^2+p_y^2)^{3/2}} & \frac{p_x}{\sqrt{p_x^2+p_y^2}} & \frac{p_y}{\sqrt{p_x^2+p_y^2}} \end{bmatrix} \tag{75}$$

If you're interested to see more details about how I got this results have a look at details below.

21.0.1 Derivation of the Jacobian H_j

We're going to calculate, step by step, all the partial derivatives in H_j:

$$H_j = \begin{bmatrix} \frac{\partial \rho}{\partial p_x} & \frac{\partial \rho}{\partial p_y} & \frac{\partial \rho}{\partial v_x} & \frac{\partial \rho}{\partial v_y} \\ \frac{\partial \varphi}{\partial p_x} & \frac{\partial \varphi}{\partial p_y} & \frac{\partial \varphi}{\partial v_x} & \frac{\partial \varphi}{\partial v_y} \\ \frac{\partial \dot{\rho}}{\partial p_x} & \frac{\partial \dot{\rho}}{\partial p_y} & \frac{\partial \dot{\rho}}{\partial v_x} & \frac{\partial \dot{\rho}}{\partial v_y} \end{bmatrix} \tag{76}$$

So all the matrix H_j elements are calculated as follows:

1.
$$\frac{\partial \rho}{\partial p_x} = \frac{\partial}{\partial p_x}(\sqrt{p_x^2+p_y^2}) = \frac{2p_x}{2\sqrt{p_x^2+p_y^2}} = \frac{p_x}{\sqrt{p_x^2+p_y^2}} \tag{77}$$

2.
$$\frac{\partial \rho}{\partial p_y} = \frac{\partial}{\partial p_y}(\sqrt{p_x^2+p_y^2}) = \frac{2p_y}{2\sqrt{p_x^2+p_y^2}} = \frac{p_y}{\sqrt{p_x^2+p_y^2}} \tag{78}$$

3.
$$\frac{\partial \rho}{\partial v_x} = \frac{\partial}{\partial v_x}(\sqrt{p_x^2+p_y^2}) = 0 \tag{79}$$

4.
$$\frac{\partial \rho}{\partial v_y} = \frac{\partial}{\partial v_y}(\sqrt{p_x^2+p_y^2}) = 0 \tag{80}$$

5.
$$\frac{\partial \varphi}{\partial p_x} = \frac{\partial}{\partial p_x} arctan(p_y/p_x) = \frac{1}{(\frac{p_y}{p_x})^2+1}(-\frac{p_y}{p_x^2}) = -\frac{p_y}{p_x^2+p_y^2} \tag{81}$$

6.
$$\frac{\partial \varphi}{\partial p_y} = \frac{\partial}{\partial p_y} arctan(p_y/p_x) = \frac{1}{(\frac{p_y}{p_x})^2+1}(\frac{1}{p_x}) = \frac{p_x^2}{p_x^2+p_y^2}\frac{1}{p_x} = \frac{p_x}{p_x^2+p_y^2} \tag{82}$$

7.
$$\frac{\partial \varphi}{\partial v_x} = \frac{\partial}{\partial v_x} arctan(p_y/p_x) = 0 \tag{83}$$

8.
$$\frac{\partial \varphi}{\partial v_y} = \frac{\partial}{\partial v_y} arctan(p_y/p_x) = 0 \tag{84}$$

9.
$$\frac{\partial \dot{\rho}}{\partial p_x} = \frac{\partial}{\partial p_x}\left(\frac{p_x v_x + p_y v_y}{\sqrt{p_x^2+p_y^2}}\right) \tag{85}$$

In order to calculate the derivative of this function we use the quotient rule.

Given a function z that is quotient of two other functions f and g:

$$z = \frac{f}{g} \tag{86}$$

its derivative with respect to x is defined as:

$$\frac{\partial z}{\partial x} = \frac{\frac{\partial f}{\partial x} g - \frac{\partial g}{\partial x} f}{g^2} \tag{87}$$

in our case:

$$f = p_x v_x + p_y v_y \tag{88}$$

$$g = \sqrt{p_x^2 + p_y^2} \tag{89}$$

Their derivatives are:

$$\frac{\partial f}{\partial p_x} = \frac{\partial}{\partial p_x}(p_x v_x + p_y v_y) = v_x \tag{90}$$

$$\frac{\partial g}{\partial p_x} = \frac{\partial}{\partial p_x}\left(\sqrt{p_x^2 + p_y^2}\right) = \frac{p_x}{\sqrt{p_x^2 + p_y^2}} \tag{91}$$

Putting everything together into the derivative quotient rule we have:

$$\frac{\partial \dot{\rho}}{\partial p_x} = \frac{v_x \sqrt{p_x^2 + p_y^2} - \frac{p_x}{\sqrt{p_x^2+p_y^2}}(p_x v_x + p_y v_y)}{p_x^2 + p_y^2} = \frac{p_y(v_x p_y - v_y p_x)}{(p_x^2 + p_y^2)^{3/2}} \tag{92}$$

10. Similarly

$$\frac{\partial \dot{\rho}}{\partial p_y} = \frac{\partial}{\partial p_y}\left(\frac{p_x v_x + p_y v_y}{\sqrt{p_x^2 + p_y^2}}\right) = \frac{p_x(v_y p_x - v_x p_y)}{(p_x^2 + p_y^2)^{3/2}} \tag{93}$$

11.

$$\frac{\partial \dot{\rho}}{\partial v_x} = \frac{\partial}{\partial v_x}\left(\frac{p_x v_x + p_y v_y}{\sqrt{p_x^2 + p_y^2}}\right) = \frac{p_x}{\sqrt{p_x^2 + p_y^2}} \tag{94}$$

12.

$$\frac{\partial \dot{\rho}}{\partial v_y} = \frac{\partial}{\partial v_y}\left(\frac{p_x v_x + p_y v_y}{\sqrt{p_x^2 + p_y^2}}\right) = \frac{p_y}{\sqrt{p_x^2 + p_y^2}} \tag{95}$$

So now, after calculating all the partial derivatives our resulted Jacobian H_j is:

$$H_j = \begin{bmatrix} \frac{p_x}{\sqrt{p_x^2+p_y^2}} & \frac{p_y}{\sqrt{p_x^2+p_y^2}} & 0 & 0 \\ -\frac{p_y}{p_x^2+p_y^2} & \frac{p_x}{p_x^2+p_y^2} & 0 & 0 \\ \frac{p_y(v_x p_y - v_y p_x)}{(p_x^2+p_y^2)^{3/2}} & \frac{p_x(v_y p_x - v_x p_y)}{(p_x^2+p_y^2)^{3/2}} & \frac{p_x}{\sqrt{p_x^2+p_y^2}} & \frac{p_y}{\sqrt{p_x^2+p_y^2}} \end{bmatrix} \tag{96}$$

22 EKF Algorithm Generalization

No equations.

23 EKF with Linear Functions Quiz

No equations.

24 Sensor Fusion General Processing Flow

No equations.

25 General EKF Algorithm

No equations.

26 Evaluating the Performance

26.1 RMSE

$$RMSE = \sqrt{\frac{1}{n}\sum_{t=1}^{n}(x_t^{est} - x_t^{true})^2} \tag{97}$$

27 13_State prediction - 4D case // T_SC_

Linear Motion - 2D:

$$\begin{cases} p'_x = p_x + v_x \Delta t \\ p'_y = p_y + v_y \Delta t \\ v'_x = v_x \\ v'_y = v_y \end{cases} \tag{98}$$

State transition:

$$\underbrace{\begin{pmatrix} p'_x \\ p'_y \\ v'_x \\ v'_y \end{pmatrix}}_{x'} = \underbrace{\begin{pmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{F} \underbrace{\begin{pmatrix} p_x \\ p_y \\ v_x \\ v_y \end{pmatrix}}_{x} \tag{99}$$

Prediction:

$$x' = f(x, u) \tag{100}$$

$$P' = F_j P F_j^T + Q \tag{101}$$

Measurement Update:

$$y = z - h(x') \tag{102}$$

28 Outro

No equations.

$$z = p' \tag{103}$$

$$z = \begin{pmatrix} p_x \\ p_y \end{pmatrix} \tag{104}$$

References