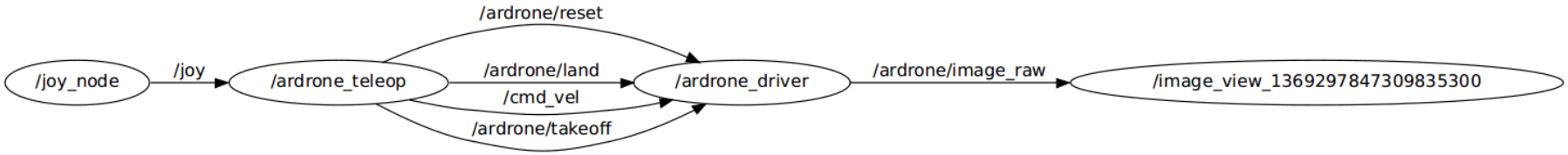
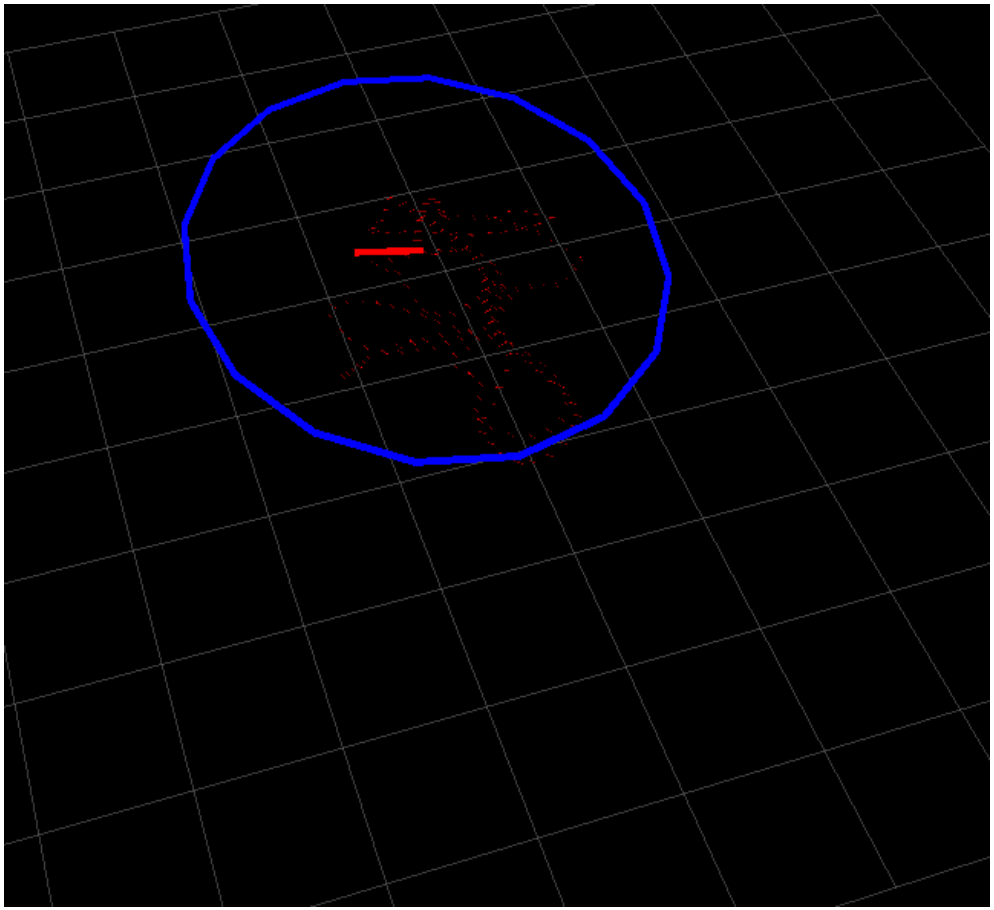
VisNav Exercise 02: Solution

Dr. Jürgen Sturm
Jakob Engel Christian Kerl

Exercise 1: Manual Flight

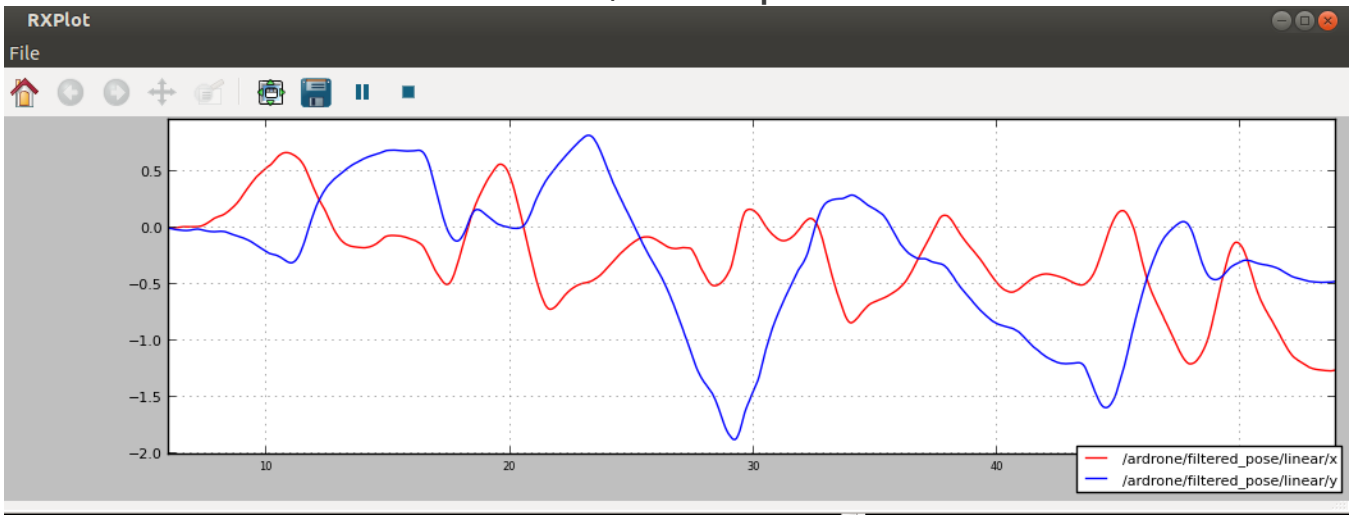


Exercise 2: EKF

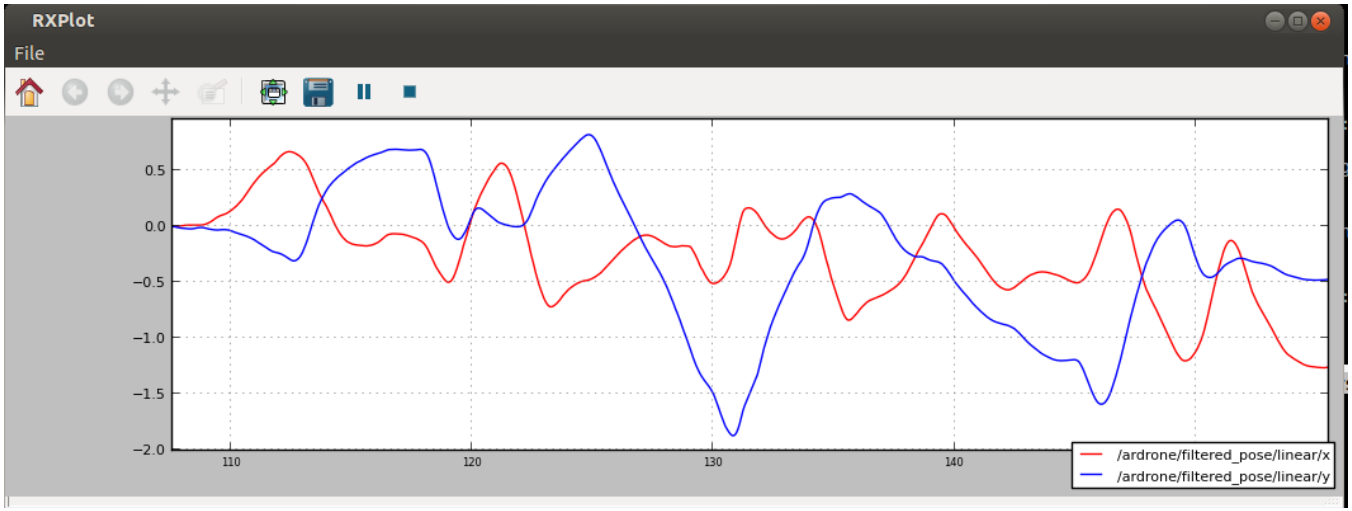


Exercise 2: EKF

No Correction, small prediction noise



No Correction, large prediction noise



Exercise 2: EKF

Measurement Function:

$$h\left(\mathbf{x} = \begin{pmatrix} x_d \\ y_d \\ \Psi_d \end{pmatrix}\right) = \begin{pmatrix} \cos\Psi_d(x_m - x_d) + \sin\Psi_d(y_m - y_d) \\ -\sin\Psi_d(x_m - x_d) + \cos\Psi_d(y_m - y_d) \\ \Psi_m - \Psi_d \end{pmatrix}$$

Drone's state

Marker Position

Measurement Function Jacobian:

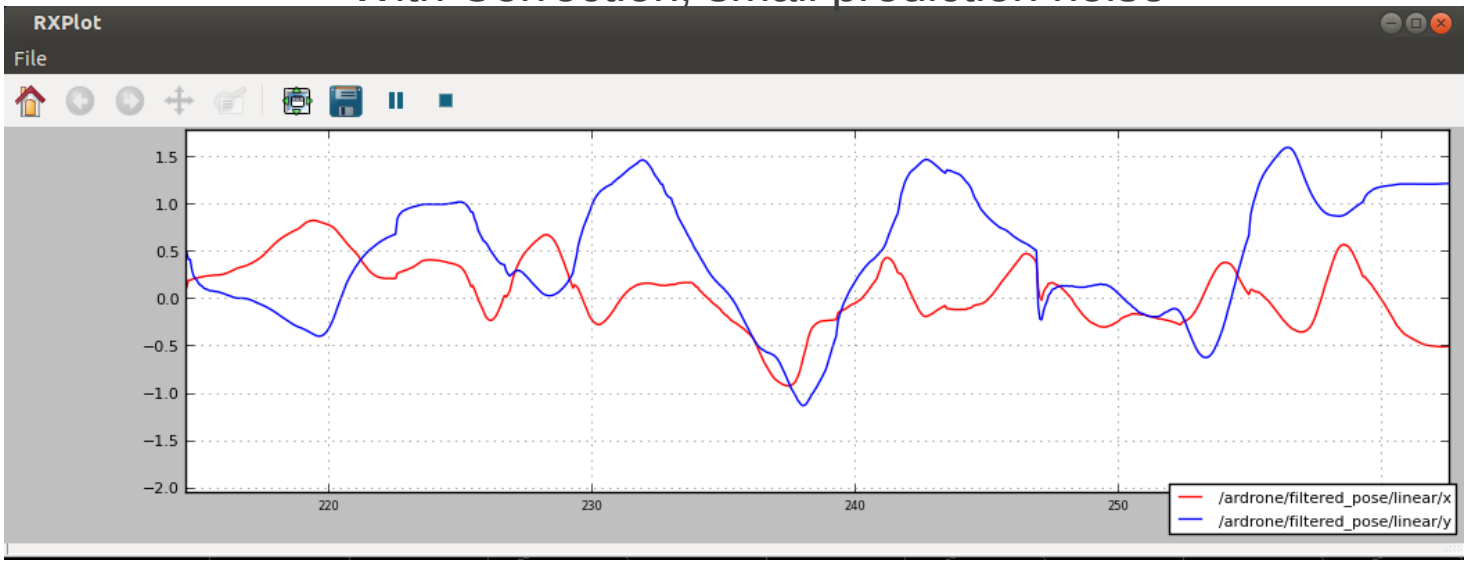
$$H = \frac{dh}{d\mathbf{x}} = \begin{pmatrix} -\cos\Psi_d & -\sin\Psi_d & -\sin\Psi_d(x_m - x_d) + \cos\Psi_d(y_m - y_d) \\ \sin\Psi_d & -\cos\Psi_d & -\cos\Psi_d(x_m - x_d) - \sin\Psi_d(y_m - y_d) \\ 0 & 0 & -1 \end{pmatrix}$$

Exercise 2: EKF

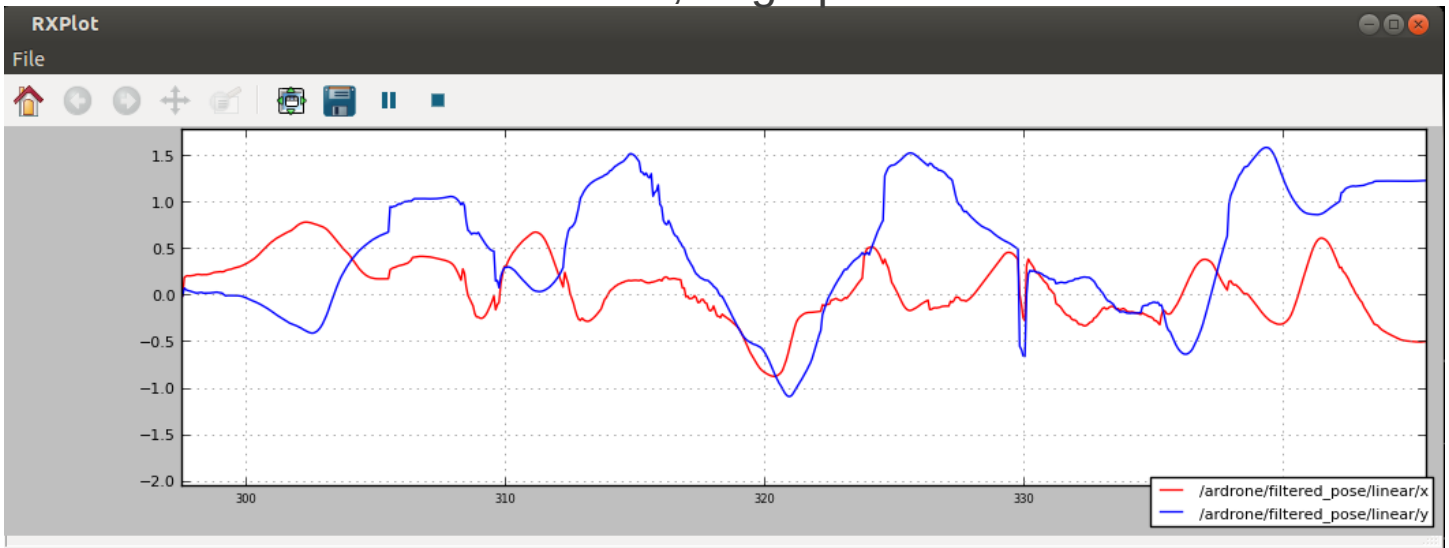
```
void ExtendedKalmanFilter::correctionStep(const Eigen::Vector3f& measurement, const Eigen::Vector3f& global_marker_pose)
{

    printf("ekf: %f %f %f;    obs: %f %f %f\n",
            state[0],state[1],state[2],
            measurement[0], measurement[1], measurement[2]);


    // compute expected measurement:
    Vector3f z_exp; // z_exp = h(x)

    float psi = state(2);

    z_exp(0) = cos(psi)*(global_marker_pose(0) - state(0)) + sin(psi)*(global_marker_pose(1) - state(1));
    z_exp(1) = -sin(psi)*(global_marker_pose(0) - state(0)) + cos(psi)*(global_marker_pose(1) - state(1));
    z_exp(2) = global_marker_pose(2) - psi;

    Vector3f err = measurement - z_exp;

    // normalize angle: diff has to be between -180 and 180 degree.
    while(err[2] > M_PI) err[2] -= 2*M_PI;
    while(err[2] < -M_PI) err[2] += 2*M_PI;

    // dh/dx
    Matrix3f H;
    H << -cos(psi),  -sin(psi),   sin(psi)*(state(0) - global_marker_pose(0)) - cos(psi)*(state(1) - global_marker_pose(1)),
         sin(psi),   -cos(psi),   cos(psi)*(state(0) - global_marker_pose(0)) + sin(psi)*(state(1) - global_marker_pose(1)),
         0,          0,           -1;


    Matrix3f K = sigma * H.transpose() * ((H * sigma * H.transpose() + R).inverse()); // Kalman Gain


    //  correct pose estimate
    state = (state + K*( err ));
    sigma = (Matrix3f::Identity() - K*H)*sigma;

}
```

Exercise 2: EKF

With Correction, small prediction noise

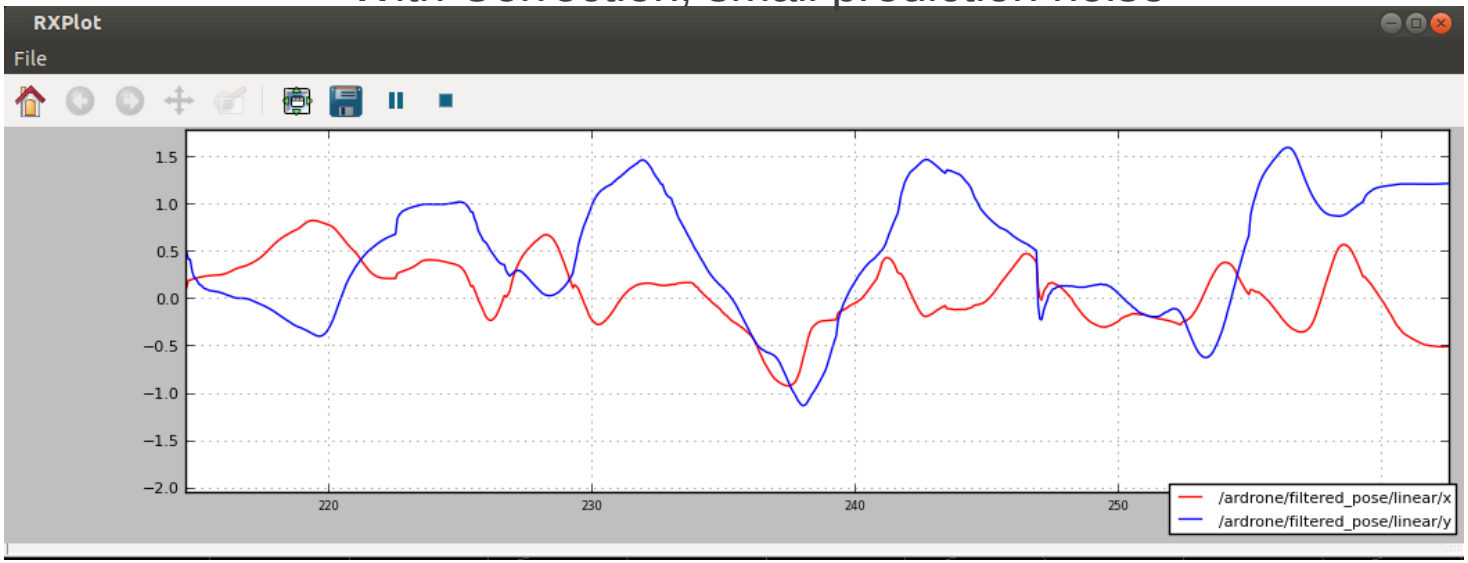


With Correction, large prediction noise

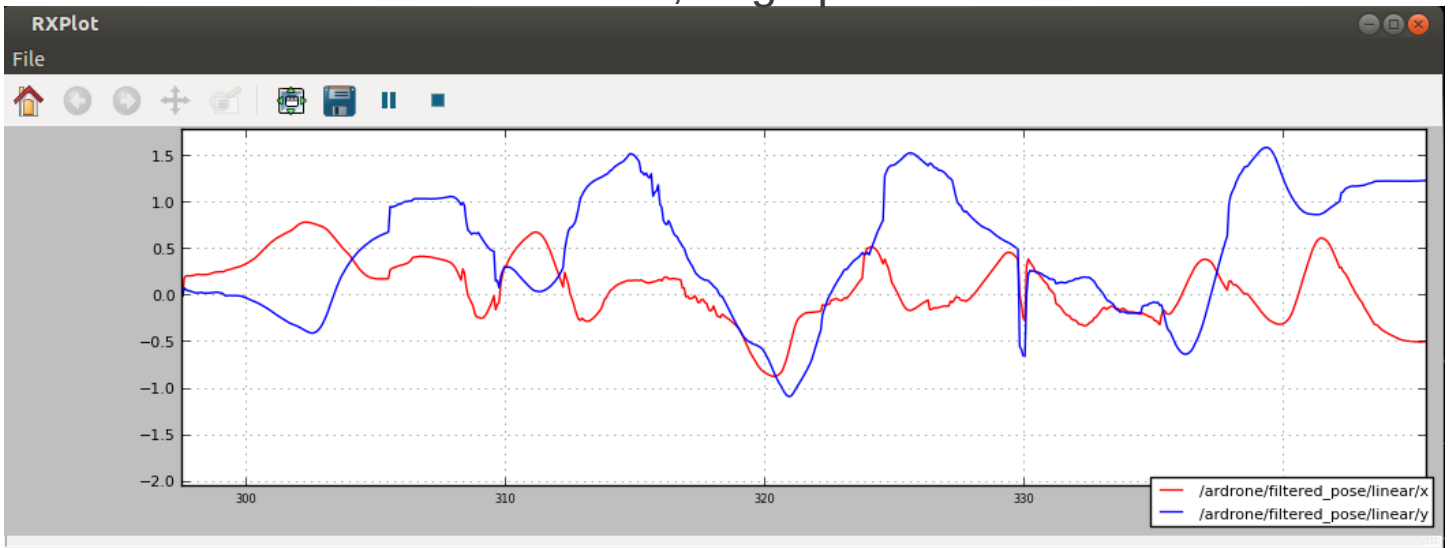


Exercise 2: EKF

With Correction, small prediction noise

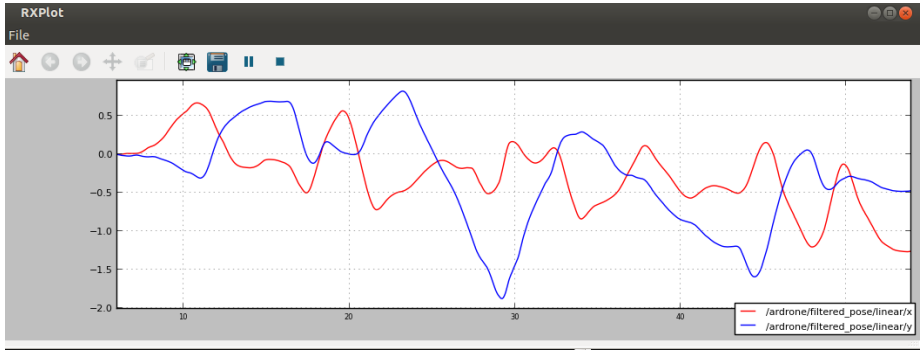


With Correction, large prediction noise

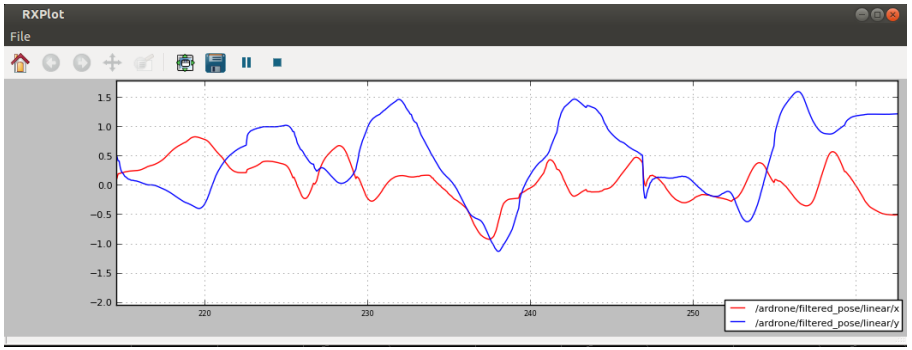


Exercise 2: EKF

No Correction



With Correction



Final x-Drift: 1.8m
Final y-Drift: 0.8m