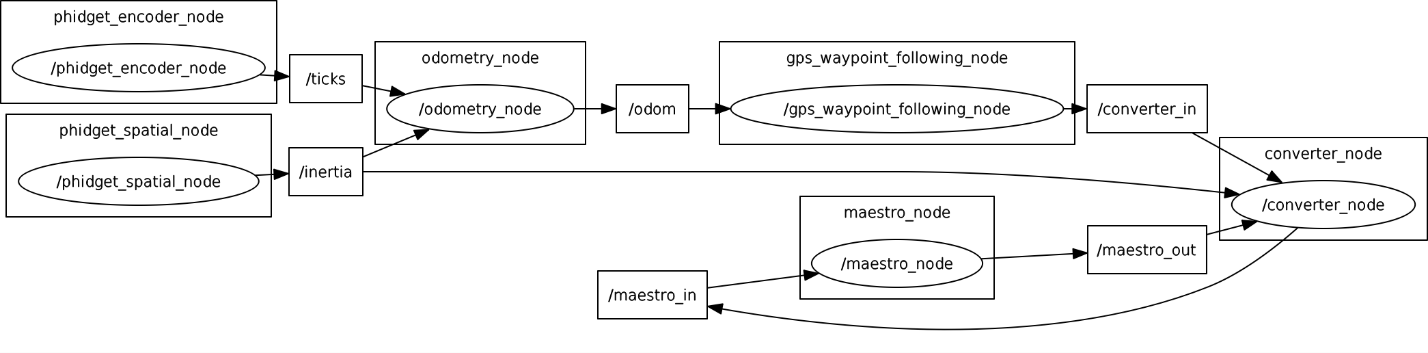
Navigation: http://kaiyuzheng.me/documents/navguide.pdf

Current Pipeline: (missing one usb port for gps)

**Phidget\_encoder\_node:**

1. Subscribe: none
2. Publish: ticks
3. Publish information: position, relative\_change, elaspsed\_time from **getTicks()** method

**Phidget\_spatial\_node:**

1. Subscribe: none
2. Publish: inertia
3. Publish information: timeSecs, timeMicroSecs, sAccel, yAccel, zAccel, xAngular, yAngular, zAngular, xMagField, yMagField, zMagField from **getInertia()** method

**Phidget\_gps\_node:**

1. Subscribe: none
2. Publish: gps\_data
3. Publish information: fix\_status, latitude, longtitude, altitude, is\_velocity\_valid, velocity, is heading\_valid, heading

**Odemetry\_node:**

1. Subscribe: ticks, gps\_data, inertia
2. spatialCallback: get the angularVelocity from zAngular
3. gpsCallback: setup latitude, longitude, altitude, heading, and update position(0), position(1), position(2) respect to easting, northing, heading **?**
4. encoderCallback: calculate the velocity in m/s
5. Publish: odom
6. Information handling:
   1. timeUpdatePhase(): update xCurr(position(0)), yCurr(position(1)), theta(position(2)), and calculating thetaPrime(theta+angularVelocity\*dt), and calculate xPrime and yPrime(eg. xCurr + (velocity \* dt \* cos/sin(theta)))
   2. projectCovariance(): update covariance matrix prediction
7. Publish information: position.x(xPrime), position.y(yPrime), position.z(0), linear.x(velocity), angular.z(angularVelocity), orientation(calculated from thetaPrime)

**Gps\_waypoint\_following\_node:**

1. Subscribe: odom, start\_command
2. odometryCallback: get the xCurr(position.x) and yCurr(position.y), and calculate thetaCurr
3. startCallback: check if data is “start” **?**
4. Publish: converter\_in, cmd\_vel
5. Information handling:
   1. sampleAndProjectControls(): try the best angVel with shortest distance to current point
6. Publish information:
   1. Odom: velocity(0.7), angVel as “angle”
   2. Cmd\_vel: linear.x(velocity), angular.z(angVel)

**Converter\_node:**

1. Subscribe: converter\_in, maestro\_out, inertia
2. targetsCallback: get the velocity and angle, and calculate desired angular velocity(ang), get drive and steering position from convertToPulsewidth() method, then publish to maestro\_in
3. pulsewidthCallback: get the drive and steering position, and get velocity, angle, and currTime from convertFromPulsewidth() method, then publish to converter\_out
4. imuCallback: get zAngular and convert it to radian as currentOmega
5. Publish: maestro\_in, converter\_out

**Maestro\_node:**

1. Subscribe: maestro\_in
2. Callback: set the drive and steering position
3. Publish: maestro\_out
4. Publish information: drive\_position, steering\_position, currTime