# 2.166 Research Assignment 1 (Tips)

Massachusetts Institute of Technology

September 9, 2008
Issued: Wednesday, September 3rd, 2008
Due: Wednesday, September 17th, 2008

Before we embark on a detailed investigation of probabilistic algorithms for robot navigation, mapping and control, this first assignment concerns the topics of robot programming using MOOS, and simple sensor processing and trajectory control.

**Problem 1.** The primary goal this assignment is to enable you to write your own MOOS applications. To get started, write a MOOS application that implements a "Waypoint Controller" that steers the robot around the (simulated or real) environment using odometry measurements for position estimation. You should write this as a new MOOS application (e.g. call it pWaypoint) that operates with the existing MOOS processes iRobotSim, pRobotViewer, iRemote, MOOSDB, and pLogger. We'll give you a demo in class of how to do this.

Detailed directions for downloading, building, and running the software are provided at:

`https://web.mit.edu/2.166/www/software/index.html`

Be sure to ask for help if you need it, by emailing the course staff.

## Implementing RA1

**Setup** Follow the instructions for downloading and building the software on the class website. Try running the software as shown on the website to ensure you're source tree is setup correctly. Note the instructions for setting up your path variable for your shell.

**Interface** In your application, you will need to register and publish the appropriate MOOS variables. Specifically, you'll need to subscribe for `ROBOT_POSITION` and publish `DESIRED_THRUST` and `DESIRED_RUDDER`. You may also choose to register for `ROBOT_X ROBOT_Y ROBOT_HEADING` if you wish to avoid parsing `ROBOT_POSITION`. Registering for `DESIRED_LIGHT` will allow you to receive waypoints from user clicks in robotviewer which marks the position with a yellow arrow shape.

```
bool CWayPoint::OnConnectToServer() {
 // ...
 m_Comms.Register("ROBOT_POSITION",0);
 m_Comms.Register("DESIRED_LIGHT",0);
 // ...
 return true;
}
bool CWayPoint::Iterate() {
 // ...
 // your waypoint control function
 if (GoToWayPoint(dfThrust,dfRudder)) {
```

```
    m_Comms.Notify("DESIRED_THRUST",dfThrust);
    m_Comms.Notify("DESIRED_RUDDER",dfRudder);
  }
  // ...
  return true;
}
bool CWayPoint::GoToWayPoint(double &dfThrust,double &dfRudder) {
  // sets value of dfThrust and dfRudder appropriately
  // could return false if no valid waypoint.
}
```

**Parsing** Within your `OnNewMail` you will receive the subscribed messages. The following procedures may help in parsing the string value of the `CMOOSMsg`.

```
bool CWayPoint::ReceiveRobotPosition(const std::string &sMsg) {
  double dfTime,dfX,dfY,dfH;
  int n = sscanf(sMsg.c_str(),
                 "time=%lf,x=%lf,y=%lf,theta=%lf",
                 &dfTime,&dfX,&dfY,&dfH);
  if (n<4) {
    // parse error
    return false;
  }
  // set current time and position
  return true;
}
bool CWayPoint::ReceiveDesiredLight(const std::string &sMsg) {
        double dfX, dfY;
  int num = sscanf(sMsg.c_str(), "%lf %lf", &dfX, &dfY);
  if (n<2) {
    // parse error
    return false;
  }
  // set current waypoint goal
  return true;
}
```

Alternatively, you may simply inline the parsing in your `OnNewMail` function.

```
bool CWayPoint::OnNewMail(MOOSMSG_LIST &NewMail) {
  // ...
  MOOSMSG_LIST::iterator p;
  for (p=NewMail.begin();p!=NewMail.end();p++) {
    CMOOSMsg &Msg = *p;
    if (Msg.m_sKey=="DESIRED_LIGHT") {
      double x,y;
      if (sscanf(Msg.m_sVal.c_str(),"%lf %lf",&x,&y)<2) {
        // parse error

      } else {
        // set your new waypoint goal
      }
    } else if (Msg.m_sKey=="ROBOT_POSITION") {
      // inline parsing here
    } else {
```

```
        // ...
      }
    }
    // ...
}
```

**Misc** In implementing your waypoint controller, you will need to steer the robot toward the goal and control the speed. You will need to account for error with a goal tolerance. Create a parameter you read from your config block in the moos file, i.e.

```
  // read in a configuration parameter
  double dfVal;
  if (m_MissionReader.GetConfigurationParam("GoalRadius",dfVal))
     m_dfGoalRadius=dfVal;
```

One easy approach to waypoint control is to first turn toward the goal and then drive toward it. If you choose this approach, be aware of hysteresis may occur around your desired heading if the gain is too high. A gain proportional to the error can avoid this (like a PD-controller), where the response is $K_d\,(h_{desired} - h_{robot})$. You can make the gain a configuration parameter in your moos file to tune without having to recompile your application.

Try to thiink about which class variables you might need to implement your waypoint controller. You should have at least variables for current pose, a goal waypoint, and whether the goal is complete or invalid (unset).