530.421 Mechatronics
Cat&Mouse MATLAB Simulation

Introduction
This small piece of code is written to help you in lab4 (I wish). You can click 'run' without changing anything in the code to see what will happen.

Visualization
The background is in black, with a red dot stands for 'Mouse' and a green dot stands for 'Cat'. The cat dot will gradually get close to mouse and finally tightly follow it.

Coordinates: grid coordinate, which is different from image coordinates. Grid coordinates is the same as what we use in most of mathematics and physics problems, x axis pointing to the right hand side and y axis pointing upwards. While image coordinates have x (row index) pointing downwards and y (col index) points to the right hand side.

State of mouse: [x, y], horizontal and vertical coordinates in grid.

State of cat: [x, y], in fact it should be [x, y, theta], theta is the direction of cat robot with respect to grid coordinate, if you choose a three wheel design (two driving wheel + caster wheel). Basically, it's an extra z axis your robot could rotate about. For convenience of simulation I ignored direction.

Algorithm:
The basic idea is using position difference between cat and mouse, update cat's position to match mouse's position.

In each step I compute the difference between mouse's position and cat's position in grid coordinates, store as a vector. This difference vector can be view as 'error' in a control problem (we want to control the difference of two object's positions to be zero). Then I applied PID control law to compute the real 'update vector', which is added to the current position of cat, to update cat's position.

Playing Notes:
You can play with Kp, Ki, Kd, and will observe different behaviors of cat robot. The current setting is good as the position of cat and mouse will match exactly after 100 iterations (0 error). But if simulation settings are changed, these three control parameters should be fine-tuned again.

Further Thinking:
This simulation is far from the task we need to tackle in lab4. Two major difficulties I find are:
-First the camera view is not perpendicular to ground, which means trying to make a grid coordinates might not be possible.
-Second, no robot is perfect, and most of the robots you build are not likely to move in omni-direction. This makes the 'vector updating' approach in simulation hard to implement. You may need to first turn your robot to match the direction of the 'update vector', and then go straight forward in that direction for some distance (**turn and move forwards**).
-Third, making a robot go straight seems like a simple job, but it is far from simple. Even if you set same PWM for both motor, your robot is much likely to move in a circle, which indicates different speed on both wheels. Again, you may need a close control loop. Maybe use a compass sensor to compensate for directional error?
-Time delay, if large, it ensures occurrence of a significant steady state error.