CTY Robotics and Applied Computing Northeastern University College of Computer and Information Science

With Funding from the National Science Foundation

Dimitrios Kanoulas (dkanou@ccs.neu.edu)
Marsette Vona (vona@ccs.neu.edu)

Contents

1 Introduction

Time to spend in this section: 20 min

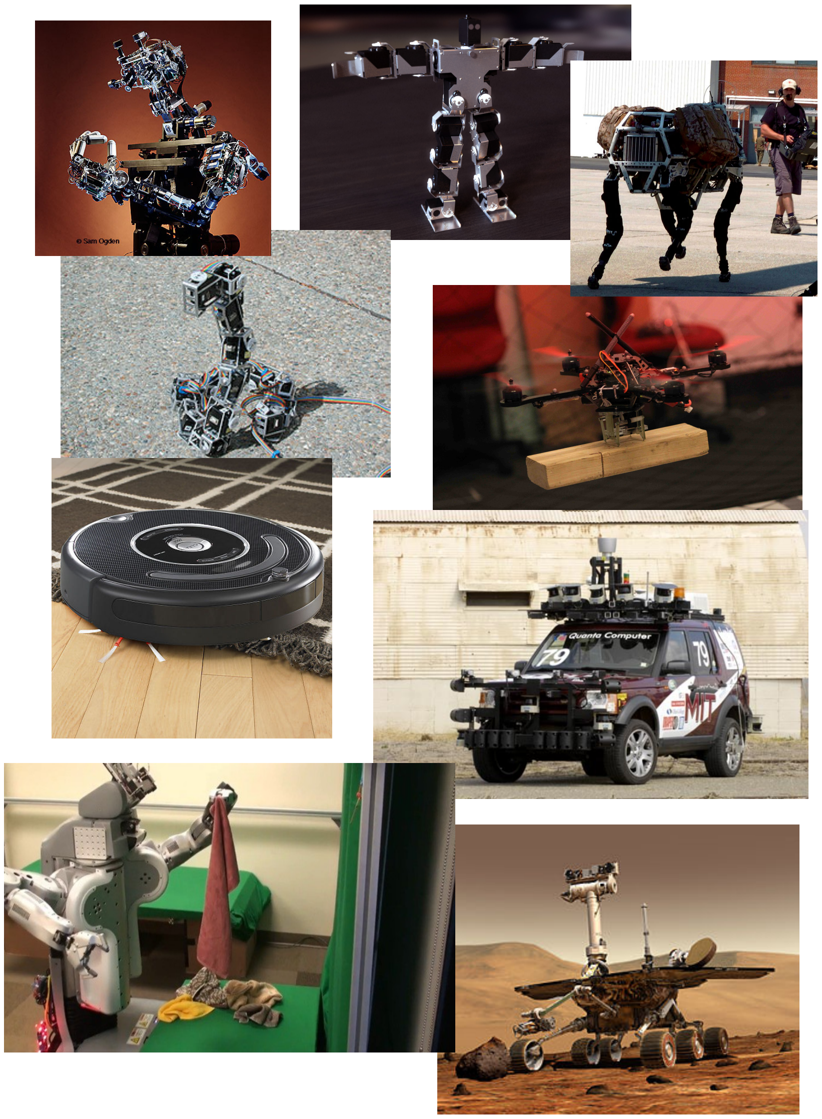


Figure 1: **First row:** COG torso with head and arms [MIT], Robo biped, and Big-Dog quadruped [Boston Dynamics]; **Second row:** snakebot [CMU], and flying quadrotor [Upenn]; **Third row:** Roomba wheeled robot [iRobot], automated car [MIT]; **Fourth row:** PR2 mobile manipulator [Willow Garage], and Mars Rover [NASA]

A **ROBOT** is a device which perceives information about the world through **sensors** (cameras, lasers, GPS, etc.) and which applies **controlled** forces with **actuators** (motors) to achieve physical **goals**.

1.1 OHMM: The Open Hardware Mobile Manipulator

The robot we will be using today is an open platform (http://www.ohmmbot.org) developed at Northeastern University for teaching robotics software. It is called the Open Hardware Mobile Manipulator (OHMM) because it can both move about in the world (mobility) and it can pick up objects (manipulation).

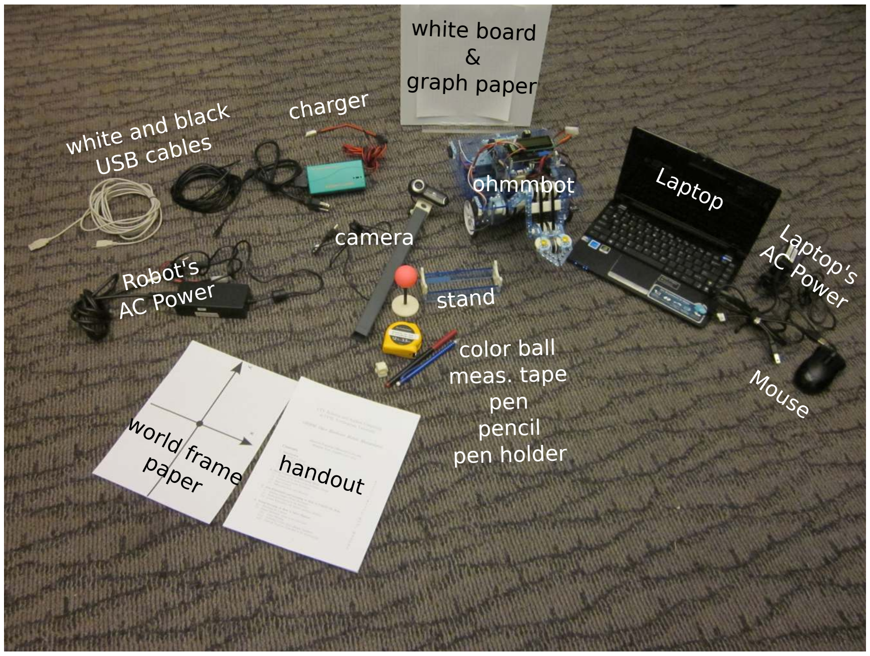


Figure 2: The Experimental Setup

OHMM is a robot with four vertical layers (see Figure 3):

- **Level 1:** Two **wheels** are controlled by separate **motors**. A **battery** (see also Figure 4, right), and an **omni wheel** (not controlled by motor) are mounted in the rear.

The motors on OHMM have sensors called **encoders** that tell us how far the motor has turned. Carefully set your robot vertically (follow the demonstration) so you can see the encoders. In these encoders, a rotating magnet passes sensors which register the rotation. Many other encoders are optical (light operated) instead of magnetic.

- **Level 2:** This level includes the **arm** with three joints, the gripper, the arm's four **motors** (one for the gripper) (Figure 4), two **bump switches** in the front, and an **IR sensor** for distance detection on the right. We will not use the bump switches or the IR sensor today.

- **Level 3 & 4:** These levels contain a high-level **processor** called a Pandaboard (from Texas Instruments) and a low-level processor called an Orangutan (from a company

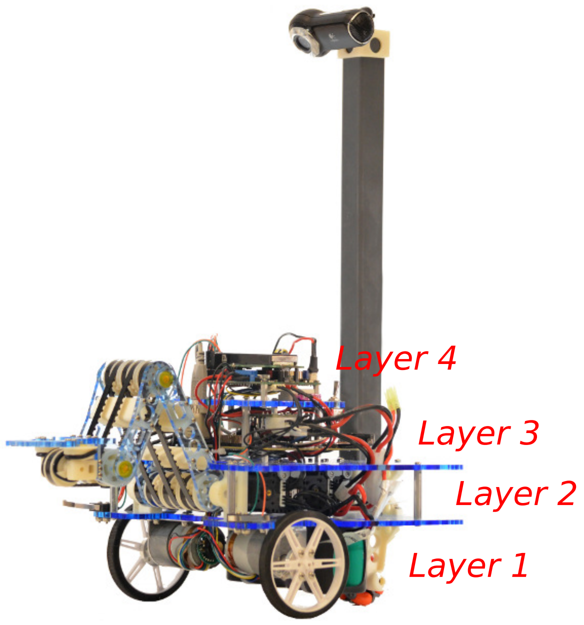


Figure 3: The OHMM Robot, a mobile manipulator for teaching robotics software.

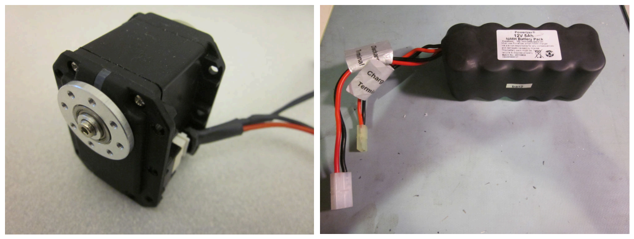


Figure 4: Motor (left) and battery (right)

called Pololu). These are small computers: the first one is powerful and it is used for tasks like image processing, and the second one is simple but reliable for motor control. We will not use the Pandaboard today. Instead we will use a **netbook** for image processing and high-level control. It will talk to the Oranguatan low-level processor with a USB link.

We will also use:

- a **USB camera** that we will mount on the robot later in Section 4.
- a **netbook** to give commands, control the robot, and display information from it (such as camera images). If you have never used **Linux** before, this is a day to remember! It is a good system for working with robots.

OHMM can:

- Move around using its **wheels**.
- Perceive objects, using the **camera**.
- Manipulate objects using its **arm** and **gripper**.

1.2 Today's Activities

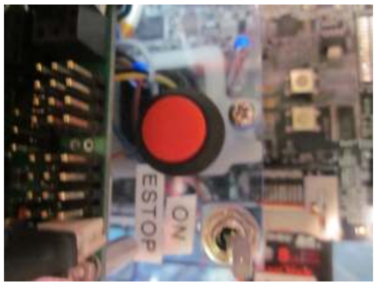
First we will learn how to control the wheels and drive the robot around. **Then** we will learn how to control the arm and place it in any position we want. **Finally**, we will use the camera to detect a pink ball, drive the robot to it, grasp it with the arm, and return "home" with it.

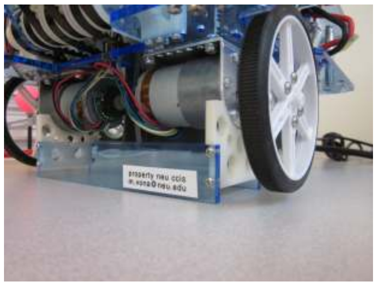
- **Section 2: Motor Control and Differential Drive** *(60 min)*: How do the wheel motors work? How do we study their motion? Let's drive the robot!
- **Section 3: Arm Kinematics and Grasping** *(60 min)*: How can we move the arm's joints and the gripper both by setting particular angles to each one of them (*forward kinematics*) or by setting a particular position and let the robot decide the joint angles (*inverse kinematics*)? Let's move the arm to any position we want!
- **Section 4: Visual Servoing** *(40 min)*: How do we use the camera to detect a brightly colored object? How do we use the differential drive and then the arm kinematics and grasping to automatically pick up the object and move it to a different place?

1.3 Safety

It is possible that things **may go wrong**

- ***When in doubt, ask for help!***
- keep your fingers away from the moving parts of the robot when it's turned on.
- Don't let hair or dangling jewelry get tangled in robot's parts (*please tie long hair back*).
- When on the table, the robot should be **on the stand at all times**. We don't want the robot to fall from the table by mistake.
- If there is a strange noise, an unexpected action of the robot, or any other emergency, push the **RED emergency stop** button on the top of the robot.





- **The robot has a number of wires. Watch carefully that these do not get caught as it moves.**

1.4 Graphical User Interface

We have created a Graphical User Interface (**GUI**) to help you give commands to the robot. Specific instructions for using it are provided in the sections below.

Be careful: Please follow the instructions when using the GUI to avoid damage to the robot.

2 Motor Control and Differential Drive

Time to spend in this section: 60 min

The first thing to try with a wheeled robot is to **drive it around.**

1. **First** we will see how to **control the speed** of the wheels.
2. **Second** we will see how we can represent the **position** and the **orientation** (i.e., the pose) of our robot on the ground.
3. **Finally** we will learn about the motion of the wheels using the **robot's kinematics**: *for example how I can drive the robot 10 cm ahead and turn it 90 degrees right.*

Don't worry if you do not understand some parts of the kinematics. We are going to learn more in the next section when we will deal with the arm.

2.1 Motor Control

We will need the **motors** that drive the wheels to rotate at a desired **speed**. Since the wheel motors are symmetric we will experiment with the **left motor** (the right one is similar).

What is Control and why is it needed? Many engineered systems (like motors) have an input which affects the value of some output. The feedback these motors give us is how much rotation has occurred. If we want to move the wheel with a desired speed $v_d = 0.5$ m/s but its current speed is $v_c = 0.3$ m/s, then a **control action** (change in motor power) should be applied to reduce the **error**

$$e = (v_d - v_c) = (0.5 - 0.3) = 0.2\text{m/s}$$

.

How to reduce the error e**?**. The most simple controller is the **Proportional Controller** (or "P Controller") which works in two stages:

1. We apply motor power proportional to the error e: $K_p e$, where K_p is a *proportional gain* constant that we can tune.
2. We should then *clamp* the output, so that we don't make too big of a change at once. The control action will be repeated very frequently by the low-level processor (hundreds of times per second).

Challenges with finding a good value for K_p**:**

- **Large values** often lead to oscillations. This can make the system go **unstable**; the oscillations can get larger and larger until something breaks!
- Some systems require **nonzero action** even when the error is zero. This includes our motor under velocity control. Observe that if we just used a proportional controller, then $e = 0$ and the control action (motor power) would be zero whenever the setpoint is reached. This would make the motor turn off!

- As long as K_p is not large enough to cause oscillation, the system will reach a steady state but it may have a **nonzero error**.

Experiment with OHMMMotorUI

For our experiments we will use a **P Controller** for the left wheel. We have already implemented most of the control program, but we leave it to you to tune K_p.

- **The robot should be on its stand on the table (see Safety Section above).**
- Press the **"OHMMMotUI"** tab on the top part of the GUI.
- In the **first row** you can set the Desired Velocity of the left wheel, by typing a new value in the **text box** and hitting the **"Run"** button. The velocity is in revolutions per second. You can always stop the wheel by hitting the **Stop** button.
- In the second row you can get the current velocity of the wheel (read from the encoder) by hitting the **"Get Current Velocity"** button.
- In the third row you can set the proportional gain K_p by typing in the text field and hitting the **"Set"** button.
- The **"Start PID Mode"** in the last row will enable a more advanced kind of controller called **Proportional-Integral-Derivative (PID) control**, that uses additional techniques beyond proportional control so that the velocity error can be brought near zero.

Let's try it:

1. Write a value in the desired velocity text box. A value around **0.5 rev/sec** is a good start.
2. Hit the **Run** button.
3. Is the wheel moving? You can check it physically, but you can also press the **"Get Current Velocity"** to get wheel's real velocity. Write down the desired and the current velocities in the box below. Hit **"Get Current Velocity"** multiple times to check if the velocity seems stable.
4. Press the **"Stop"** button.
5. Keep the desired velocity fixed and start increasing the proportional gain by **0.2**. Each time press the **"Stop"** button, type in the new gain, press the **"Set"** button, then press the **"Run"** button, and use **"Get Current Velocity"** update the velocity from the encoder. Write down the new current velocity.
6. Press the **"Stop"** button when the gain is **1.4**.

	Exp1	Exp2	Exp3	Exp4	Exp5	Exp6	Exp7	Exp8
P Gain	0.0							
Desired Veloc.	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
Current Veloc.								
Stable?								

We can notice two things: 1) the error between the desired and the current velocity decreases when the gain increases, and 2) if the gain is too big then the motion is unstable.

The P controller is simple, but not always the best. Although you could use one of the stable solutions you found (for example 0.8 gain might be a good option), to really reduce the error we are going to use a more advanced **Proportional-Integral-Derivative (PID) Controller**. Press the **"Start PID Mode"** and then the **"Run"** button. Check the current velocity now. Is it better?

2.2 Position of the Robot using Coordinate Frames

Now that we have some experience controlling the speed of the wheels, we can start to drive the robot around. First, we need to explain how we represent the **pose** of the robot on the ground. (Don't put your robot on the ground just yet.)

- We will call the starting pose of the robot **world reference frame** (see Figure 5, upper left part).
- Then we need just **3 numbers** to describe the pose of the robot: 2 for its (x,y) **position** on the ground (in meters) , and 1 for its **orientation** θ (in degrees).

Some examples are given in Figure 5.

2.3 Forward and Inverse Kinematics

Kinematics *is the study of the geometry of motion without considering its causes (physical forces).*

We can rotate the wheels at speeds $\dot{\phi}_l$ and $\dot{\phi}_r$ respectively. But how fast does this make the robot move? This is a question of **forward velocity kinematics**: we know the motions of the actuators; we want to predict the motion of the robot. We can also ask the opposite question. We know the motion of the robot, what is the motion of the motors? This is an **inverse velocity kinematics** question.

Somewhat more useful here is the **forward position kinematics**: given a starting pose of the robot and the wheel rotation for a specific amount of time, what is the final position of the robot? We are not going to go into details but there are formulas to keep track of the robot pose in small time intervals until it arrives in the final pose. The opposite question is what we often really want to know when driving a robot: how can we move the wheels (i.e. what kind of rotational velocities should be applied) to move the robot from one pose to another?

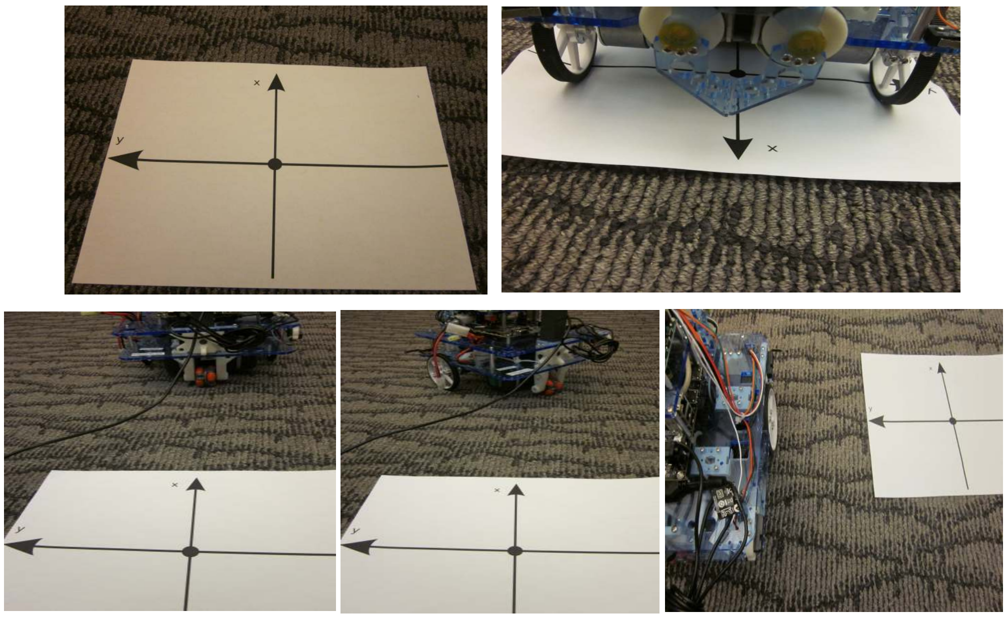


Figure 5: **Upper Left:** the world reference frame, **Upper Right:** robot on the starting position (x,y)=(0,0) and $\theta = 0$, **Lower Left:** at position (x,y) = (10cm,0) and orientation $\theta = 0$ degrees, **Lower Mid:** at position (10cm,0) and orientation $\theta = 45$ degrees, and **Lower Right:** at position (0,10cm) and orientation $\theta = 0$ degrees.

This **inverse position kinematics** question has a lot of answers, because there are many ways to move a robot from one pose to another. We are going to use a very simple one, which has only two steps:

1. turn in place to face the goal location
2. drive to the goal location

2.4 Differential Drive and Odometry

Odometry: *if we assume the wheels do not slip on the ground, we can keep track of the robot pose just by monitoring the wheel rotations. This process, which gives us an estimate of the robot's position at any time, is called **odometry**.*

Now we have a good PID Controller for our robot. It seems that when the wheel is on the air the desired speed is reached in a stable way. Let's try to give a set of commands to drive your robot in a equilateral triangle of side size 0.5m.

Experiment with OHMMDriveUI

- We will use the **measurement tape** for this experiment.
- **Place carefully the robot on the ground on its starting pose** (see upper right part of Figure 5)
- Press the **"OHMMDriveUI"** tab on the top part of the GUI.
- The **Commands Part** (Upper Left) of the GUI has 5 starting command lines. Each line has: a pulldown that says **noCmd** (No Command) and a text box (**0.0**). Click on noCmd—you can select between:
  - **noCmd**: No Command
  - **df**: drive either forward if number is positive, or backwards otherwise (in meters).
  - **dt**: turn left if number is positive, or right otherwise (in degrees).

  The text box for **noCmd** is not considered, for **df** it should be the distance in **meters**, and for **dt** it should be the **degrees** of turning.
- The upper right part shows the (x,y) position in meters and the orientation in degrees, using **odometry**, after pressing the **"Get odometry"** button (see below). This pose is where the robot **thinks** it is.
- The bottom part are the running buttons:
  - **Add New Command:** adds a new command line if needed.
  - **Run Commands:** Run the commands in order.
  - **Stop:** stop the robot and reset odometry.
  - **Get odometry**: updates the odometry from the robot.

Let's try to drive in a equilateral triangle of side length 0.5m:

1. Press the **"Reset Robot"** button and then **"Get Odometry"**. Repeat until you get zeros for the odometry.
2. The robot is on the first vertex of the triangle: position $(x, y) = (0, 0)$ and orientation $\theta = 0$. Let's drive to the second vertex. Pick the correct commands to drive to the second point, and hit the button "Run Commands".
3. What is the desired (x, y) position of the second vertex? Where does the robot think it is now thatit should be there (use the odometry button)? Use the measurement tape to estimate the (x, y) position that the robot actually is. Write these values to the boxes below.
4. Replace the commands to **turn and drive** to the third vertex and write the new values to the box below.
5. Replace the commands to **turn, drive, and turn** back to the starting vertex in the orientation the robot started. Again write the values to the box below.

6. Now write a single sequence of commands that drives around the whole triangle and returns to the starting pose.

If by mistake you pick the wrong commands, hit the **“Reset Robot”** button, until you get zero odometry and start over.

Pose	**Vertex 1**	**Vertex 2**	**Vertex 3**
Goal			
Odometry			
Measure			

We can see that even after setting a good controller for the motors, the goal pose, the pose that the robot thinks it is at, and the pose that the robot really is at are all different (even slightly). This can be due to a variety of effects: friction with the ground, wrong motor sensor measurement, sloppy control, etc. This is a difficult problem to resolve.

3 Arm Kinematics and Grasping

Time to spend in this section: 75 min

The position of the arm can be described with 3 values: angles $\theta_0, \theta_1, \theta_2$ (see Figure 6; noted as t_0, t_1, t_2 below and in the GUI, in degrees), that control the shoulder, elbow, and wrist of the robot correspondingly. In this section we will learn how to move the arm of the robot and the gripper, using forward and inverse kinematics:

- **Forward Kinematics:** How does the gripper move when we change the t_0, t_1, t_2 values (the joint angles)?
- **Inverse Kinematics:** What should be the values of t_0, t_1, t_2 if we want to move the gripper to a specific position?

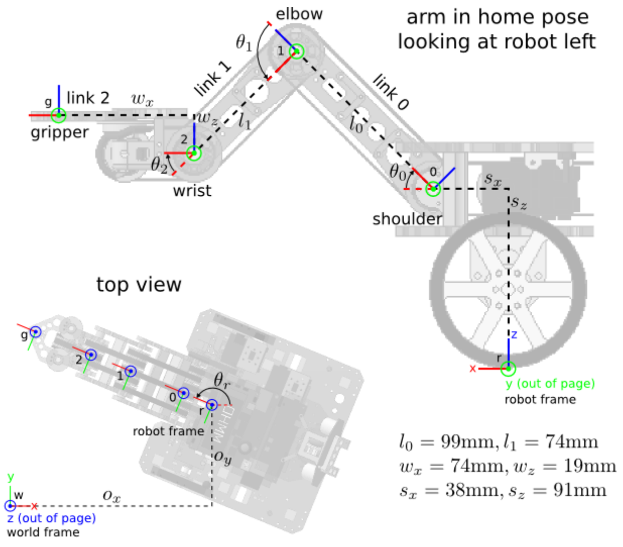


Figure 6: Arm angles and robot reference frame

3.1 Keeping Gripper (Wrist) Horizontal

Our arm can move only **up-and-down**, since the axes of the shoulder, elbow, and wrist joints are all horizontal. We can think of their combined effect on the gripper pose in three components: (1) **the height** h of the gripper along the world-frame z axis (which aims up from the ground right between the wheels), (2) the **radial distance** r out from the z axis to the gripper, and (3) the **pitch angle** ρ of the gripper about the y axis of the robot frame.

Let's move the arm:

1. Make sure you have selected the "**OHMMArmUI**" tab in the GUI at the top of the window.
2. Hit the "**Home**" button in the GUI to place the arm to the home position.

3. Now type in a value in the t_0 box in degrees, for example $t_0 = -20$ and hit **Move**.
4. Then return the joint to its home position by restoring the original angle (the home angles are $t_0 = -45$, $t_1 = 90$, and $t_2 = -45$).

Do the same for each of the other joints, but only move one at a time (and then return it to the home angle) so that you can see the effect of each joint separately.

As a simplifying design choice, we are going to control the arm in such a way that ρ is always zero, keeping the wrist horizontal and the gripper flat and parallel to the ground. One way to do this is to continuously have the computer calculate a value for t_2, given the current values of t_0 and t_1, that keeps ρ zero. **What is the formula that the computer should calculate?** Look carefully at the definition of the joint angles in Figure 6 and write the formula in the box below. Your formula can and should include the variables t_0 and t_1. Use a pencil in case you want to change your answer.

$$\boxed{t_2 = \qquad\qquad\qquad\qquad\qquad\qquad}$$

Now you'll use the GUI to have the computer do this calculation automatically. Home the arm, and then type your formula into the box for t_2. You can type the formula directly—the GUI will understand symbols like `+`, `-`, `*`, and `/`, as well as variable references like `t0` and `t1`; for example t2: $2 * t1$. Now type in some new angles for t_0 and t_1 and hit "**move**". Does the wrist remain (essentially) horizontal? If not, check your formula. In emergency either hit the **"Stop Arm"** button, or the **red button** on the robot.

3.2 Forward Kinematics & the Jacobian

With ρ fixed at zero, and t_2 slaved to t_0 and t_1, we can think of the arm as a mathematical function, or *mapping*, that takes two inputs—the joint angles t_0 and t_1—and produces two outputs: the gripper height h and the radial distance r. This is called the *forward kinematic mapping*, or just the *forward kinematics*, of the arm.

It is possible to derive mathematical formulas which compute h and r given t_0 and t_1. They involve the link lengths l_0, l_1 (see Figure 6) and trigonometric functions (sine and cosine) of the joint angles. Here, we will use an alternate method that does not require such a derivation. For both joint angles t_1 and t_2 in turn, we will

- make a small "tweak" to a joint angle
- measure the effect of that tweak on each of the two output variables h (the height of the gripper above the ground) and r (the extension of the gripper away from the robot).
- divede each "effect" by the "tweak" to *estimate a derivative*[1].

[1]If you have never learned about derivatives, which are taught in calculus, don't worry. In this handout you can substitute the word "sensitivity" for "derivative", as in "dh/dt_0 is the sensitivity (or derivative) of the gripper height with respect to change in angle of t_0: dh/dt_1 is the ratio of change in h to change in t_1."

By tweaking each of the two joints t_1 and t_2 and measuring the effect of each tweak on both h and r, we calculate four derivatives. For example, dh/dt_1 is the derivative of the gripper height with respect to motion of the elbow, and dr/dt_0 is the derivative of the radial distance with respect to motion of the shoulder. Note: the numeric value of these derivatives generally depends on the initial pose of the arm, but they can be estimated for any pose.

Let's try it. Let's measure a few of the derivatives manually using the arm. We are going to work with t_1, t_2, h, and r. That gives us four derivatives to measure, and you're going to arrange them in a 2x2 matrix[2]:

$$J = \begin{bmatrix} dr/dt_1 & dr/dt_2 \\ dh/dt_1 & dh/dt_2 \end{bmatrix}.$$

This is called a *Jacobian matrix* after the nineteenth-century mathematician Carl Gustav Jacob Jacobi (no kidding).

1. **"Home"** the arm and **retype** the equation for t_2 you found above.
2. Get the pen (don't take the cap off yet) and put it in the gripper as shown in Figure 7. Follow the instructions below the figure to grab the pen.
3. Once the pen is correctly gripped, take off the cap.
4. Now, **being careful not to let the pen touch the paper yet**, slowly move the graph paper board so that the plastic base is parallel to the robot's side.
5. Place the robot is such a position so that the pen is almost, but not quite, touching the paper.
6. **Give the paper a slight and gentle push from the back** so that it briefly comes in contact with the pen. The longer you hold it, the larger the mark the pen will make. Just make a small mark, because later you'll be measuring it's position and you want to be able to easily find the center.
7. You should have one mark on the paper at this point. Use your pencil to label it "o" for **"origin"**.
8. You'll now move t_0 and t_1 small amounts, relative to their current (home) positions, and make two additional marks.
    (a) First move t_0: leaving t_1 at its current settings, type in a new value for t_0 that moves it by a small angle, say +10 degrees, relative to its current position. The current value for t_0 should be -45. So type in the box "-35".
    (b) Hit **"Move Arm"** and then gently push the paper again to make a mark.

[2]Never studied matrices? Don't panic. All you really need to know is that a matrix is a table of numbers. All the other mathematics will be given to you, and it only involves basic addition, subtraction, multiplication, and division.

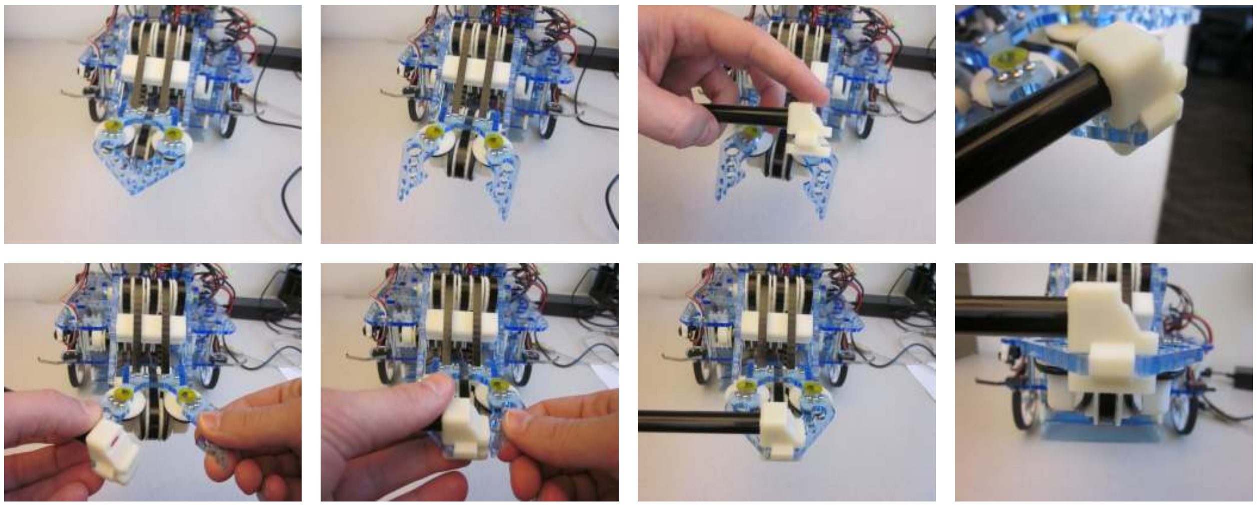


Figure 7: **Pen Holder** steps. **Step 1:** Arm is in Home position and t_2 angle has the correct equation to keep it horizontal. **Step 2:** Hit the **"Open Gripper"** button. **Step 3:** place the pen in the pen holder if it is not placed already (keep the cap of the pen on for now). **Step 4:** place the pen holder as shown in the upper right image. Hint: the grippable part of the holder is wedge-shaped; put the wider part on the outside of the gripper. **Step 5-6:** Hit the **"Close Gripper"** button and carefully adjust the holder so that it is as shown in lower right image.

(c) Label the new **mark "1"** with your pencil.

(d) Now return t_0 to its original setting of -45, and hit **"Move Arm"** again.

(e) Finally, do the same procedure for t_1 (from 90 to 100 degrees), labeling the third **mark "2"**

9. Carefully move the paper away from the pen without making any new marks.

10. **Put the cap back on the pen**, and lay the board flat on the table with the paper face up (don't pull it off the board; just let the base hang off the side of the table).

11. Use the ruler to measure (**in centimeters**) the horizontal and vertical coordinates of marks labeled "1" and "2" relative to the mark labeled "o". That is, consider "o" to be the origin of a coordinate system with h upp and r horizontal (positive going out from the robot) and label the coordinates of the other two points.

12. Divide these coordinates by the delta joint angle (10 degrees) to fill in the Jacobian matrix below. The vertical coordinates are changes in the gripper height h relative to the start pose, and the horizontal coordinates are changes in the gripper radial position r relative to the start pose. So if your points have coordinates (r_1, h_1) and (r_2, h_2) relative to the origin at point "o" you will fill in the matrix with the values of four divisions (remember it's easy to divide by 10, just move the decimal place one position to the left):

$$J = \begin{bmatrix} dr/dt_0 & dr/dt_1 \\ dh/dt_0 & dh/dt_1 \end{bmatrix} = \begin{bmatrix} r_1/10 & r_2/10 \\ h_1/10 & h_2/10 \end{bmatrix} = \begin{bmatrix} \boxed{} & \boxed{} \\ \boxed{} & \boxed{} \end{bmatrix}.$$

13. The Jacobian can be used to predict the expected gripper motion that would occur due to a small motion of the joint angles relative to the current pose. The inputs in this case are two numbers et_0 and et_1 giving the relative joint motion, and the outputs are two numbers er and eh giving the relative gripper motion. The computation works like this:

$$\begin{bmatrix} er \\ eh \end{bmatrix} = J \begin{bmatrix} ej_1 \\ ej_2 \end{bmatrix} = \begin{bmatrix} dr/dt_0 & dr/dt_1 \\ dh/dt_0 & dh/dt_1 \end{bmatrix} \begin{bmatrix} et_0 \\ et_1 \end{bmatrix}$$
$$er = (dr/dt_0)et_0 + (dr/dt_1)et_1$$
$$eh = (dh/dt_0)et_0 + (dh/dt_1)et_1$$

You won't use this formula directly today, but it leads us to a more interesting formula which computes the *reverse*: et_0 and et_1 given er and eh—a change in joint angles that will realize a desired change in gripper position.

3.3 Inverse Kinematics: the Inverse Jacobian Method

As you may have already learned, it is generally possible to *invert* a matrix[3]. For a 2x2 matrix

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

the formula for computing the inverse matrix A^{-1} (also 2x2) is

$$A^{-1} = \begin{bmatrix} d/(ad-bc) & -b/(ad-bc) \\ -c/(ad-bc) & a/(ad-bc) \end{bmatrix}.$$

The inverse of the Jacobian is exactly what we need in order to solve for et_0 and et_1 given a desired er and eh:

$$\begin{bmatrix} et_0 \\ et_1 \end{bmatrix} = J^{-1} \begin{bmatrix} er \\ eh \end{bmatrix}$$

Above, you computed the four entries of the Jacobian J for this arm at its home position. Now use the 2x2 matrix inversion formula to compute the entries of the corresponding inverse Jacobian (hint: apply the four formulas given above for a general 2x2 matrix inverse A^{-1}):

[3]Actually, not all matrices can be inverted directly. To be invertible, the matrix must be square (same number of rows as columns) and must also satisfy a second more subtle property that depends on its numeric values. It is possible for a Jacobian matrix to fail one or both of these conditions, in which case the robot is said to be at a *singular configuration*. This is interesting, but we don't have time to go into the details here.

$$J^{-1} = \begin{bmatrix} \boxed{} & \boxed{} \\ \boxed{} & \boxed{} \end{bmatrix}.$$

Let's pick a small relative gripper motion, say $er = 2$cm and $eh = -1$cm, compute the corresponding relative joint motion, and use the result to actually move the arm.

$$\begin{bmatrix} et_0 \\ et_1 \end{bmatrix} = J^{-1} \begin{bmatrix} 2 \\ -1 \end{bmatrix} = \begin{bmatrix} \boxed{} * 2 + \boxed{} * (-1) \\ \boxed{} * 2 + \boxed{} * (-1) \end{bmatrix} = \begin{bmatrix} \boxed{} \\ \boxed{} \end{bmatrix}.$$

1. Take the cap back off the pen and return the paper back so that the pen is aligned over the origin point you marked previously.

2. Type in expressions that move t_0 and t_1 relative to their current position by adding your computed et_0 and et_1 (for example, if the current value of t_0 in the GUI is -45 and you computed 10.2 for et_0, you could type in "-34.8"), hit the **"Move Arm"** button, and make a <u>final (fourth total) mark</u>.

3. Carefully move the paper away again, **put the cap back on the pen**, take the pen out of the gripper, and measure the coordinates of your new mark.

Remember, the goal was to move the gripper 2cm out radially and 1cm down from the origin at point "o". How did you do?

Though it's laborious to manually invert Jacobians, the computer can do it easily. This is one way to convert a desired motion of the gripper to the required motions at the joints.

4 Visual Servoing: Find the Ball, Drive to it, and Pick it Up!

Time to spend in this section: 25 min

Now we are going to put together the work we did in the previous sections and make the robot detect a pink ball, drive to it, pick it up, and drive it back home. This is an example of *visual servoing*, where we use measurements from a video image to continuously drive a robot to satisfy some goal. We have implemented the most of the code, but you can try to tune the ball detection using the GUI.

4.1 Attaching the Camera

It is time to use the **camera**! We need to connect the camera to the robot. Follow the steps as shown in Figure 8:

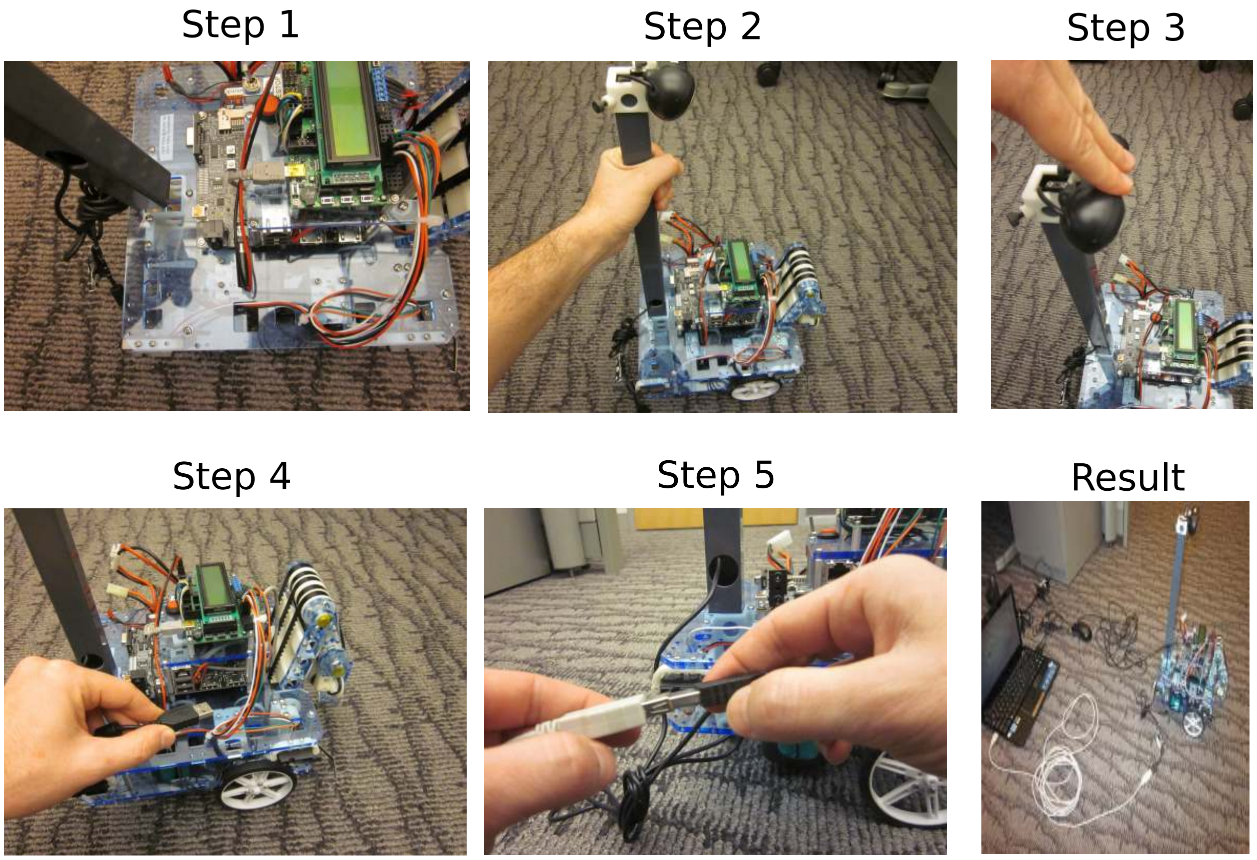


Figure 8: Attaching the Camera

1. Insert the **mast** into the mast bracket above the rear wheel. Make sure that the round hole on the mast is at robot's right side. (**Step 1** and **Step 2**)

2. Push the mast down gently. (**Step 3**)
3. Connect camera's USB cable to the **white USB cable**, which will be connected to the laptop. (**Step 4** and **Step 5**)

Now you should be able to see the camera image in a separate window by clicking on the **OHMMCameraUI tab** and then **"Init"**. You will also see the **arm** moving to a starting position. Let us know if you cannot see the image!

4.2 Visual Servoing

1. Place the robot on the ground.
2. Press the **"Reset"** button.
3. **Calibration:** Place the ball object in the gripper. Then click on the video in the middle of the ball to set the nominal color for the color filtering (you can keep clicking until it looks ok), and also to say to the robot where the ball should appear in the camera when it can grab it.

4. Place the ball in another place (but it should still appear in the video) and press the **"Run"** button. See if the robot goes and grab the ball.

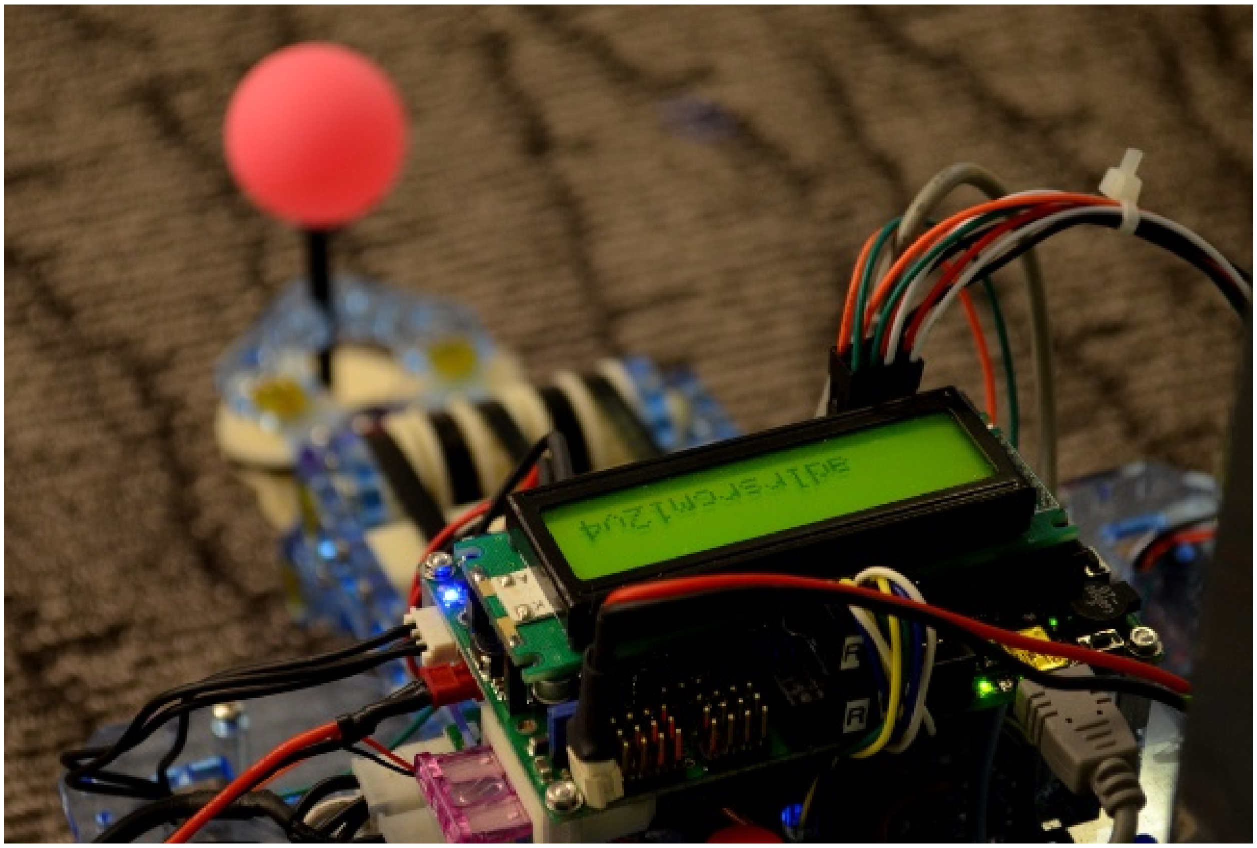
When you click Run, the robot makes small turning and driving motions to try to get the ball back to the place in the image where it was when you clicked on it. Once it gets there, it knows that it can reach out in the same way to grab it again. This is called **Visual Servoing**.

4.3 Object Detection: where is the pink ball?

The ball is detected based on its color. However, other colors in the image could be similar (like your hand). The control software uses a threshold on the **hue** (the essential color) of the ball, but setting this threshold can be tricky.

Let's try changing it:

- Type in the hue threshold text box a value between **0-255** and press **"Set"** to adjust the threshold. Start with small values and start increasing until you get a good detection. If you increase it a lot further (100 or more) then you will start to see a lot of "false positives" where the system thinks it sees the ball color.



Hope You Had Fun!