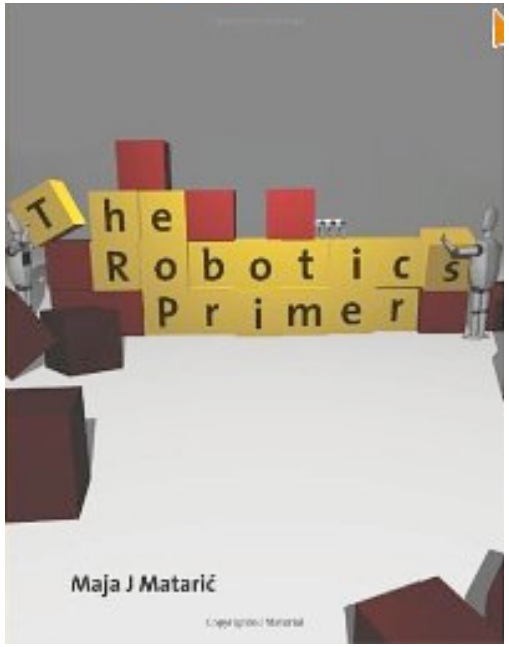
CAP4630 Artificial Intelligence Lecture 8: Robotics using Botball

Dr. Ching-Hua Chuan

February 6, 2017

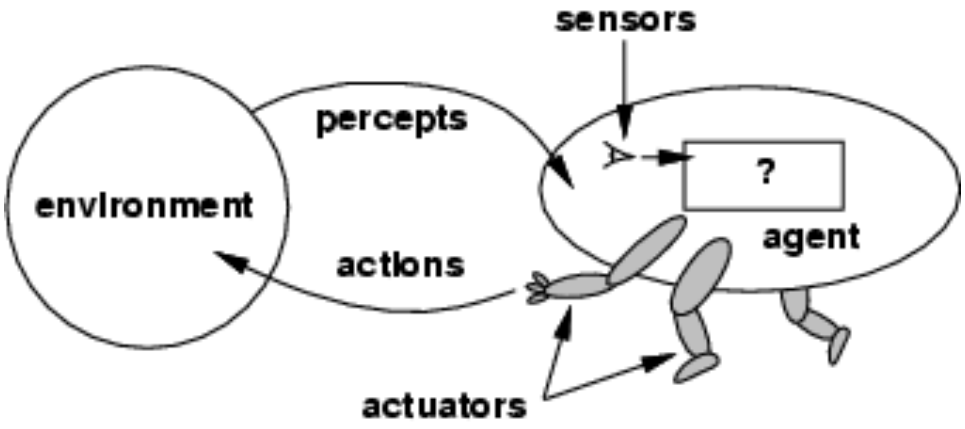
Outline

- Basic Robotics
- Botball
- AI Botball tournament



Robotics

- Robots are physical agents that perform tasks by manipulating the physical world.
  - Effectors: legs, wheels, joints, and grippers.
  - Actuators: the mechanism that enables the effector to execute an action or movement.
  - Sensors: camera, sound sensors, light sensors…



Three Primary Categories

Manipulator

Mobile Robot

Humanoid Robots

Robot Hardware: Motors

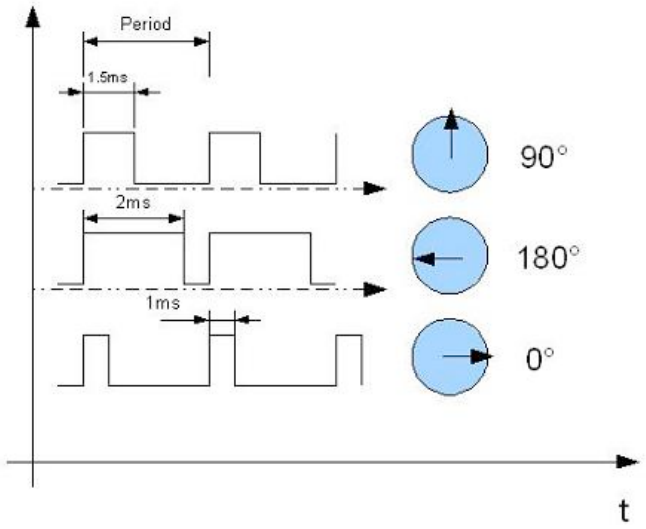
- Motors are the most common actuators in robotics.
  - Well suited for actuating wheels
- Types
  - Direct-Current (DC) motors
  - Servo motors

DC Motors

- DC motors convert electrical energy into mechanical energy.
- DC motors are great at rotating continuously in one direction.
- To run a motor, you need to provide it with electrical power in the right voltage range.
  - Too low: doesn't run or runs but has less power
  - Too high: the power will make it break down sooner

Servo Motors

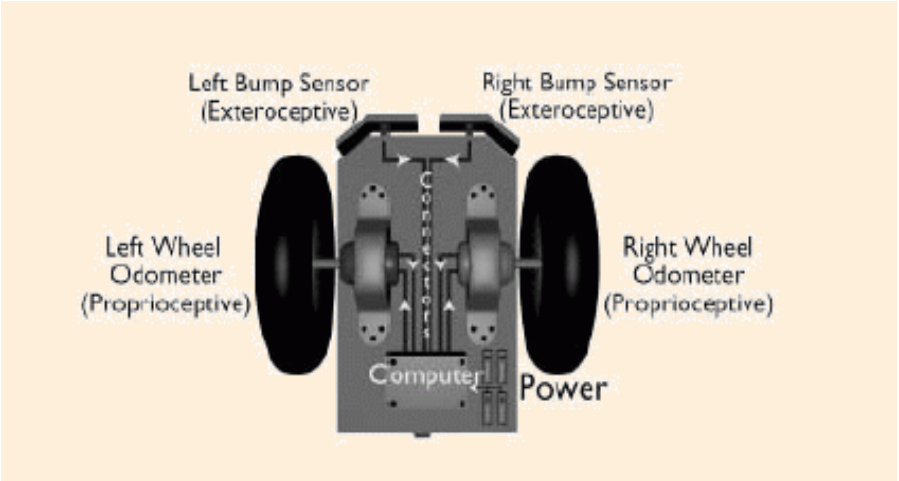
- Motors that can turn their shaft to a specific position are called servos.
- This position is somewhere along 180 degrees in either direction from a reference point.
- The amount of the motor shaft turns when the pulse arrives is determined by the duration of the pulse.



Robot Hardware: Sensors

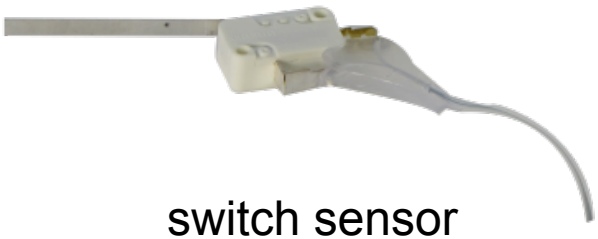
- A robot typically has two types of sensors based on the source of the information it is sensing:
  - Proprioceptive sensors: for sensing internal states.
  - Exteroceptive sensors: for sensing external states.

- Sensors do not provide states. They provide raw measured quantities.



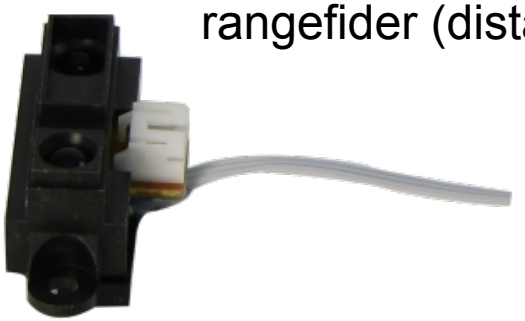
Robot Hardware: Sensors

Physical Property	Sensing Technology
Contact	Bump, switch
Distance	Ultrasound, radar, infra red
Light level	Photocells, cameras
Acceleration	Accelerometers and gyroscopes



switch sensor

rangefider (distance) sensor



light sensor

small touch sensor

Challenges in Knowing the Environment

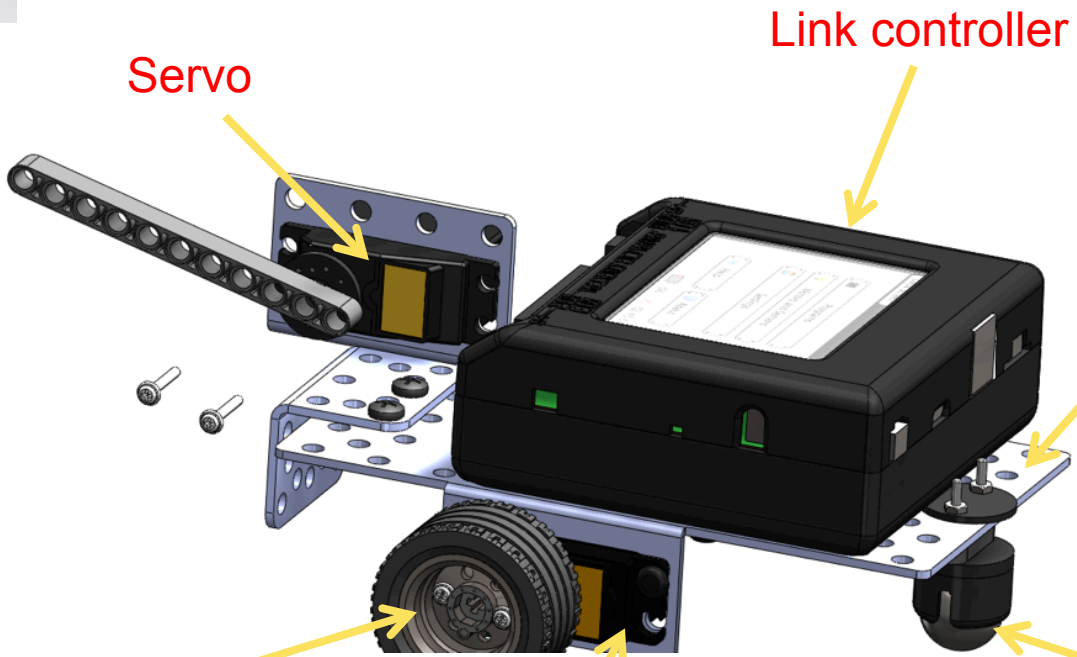
- Sensor noise and errors
- Sensor limitations
- Effector and actuator noise and errors
- Hidden and partially observable states
- Lack of prior knowledge about the environment
- Dynamic and changing environments

Botball

- Botball: standards-based educational robotics program
- http://www.botball.org/
- The KISS Institute for Practical Robotics
- http://www.kipr.org

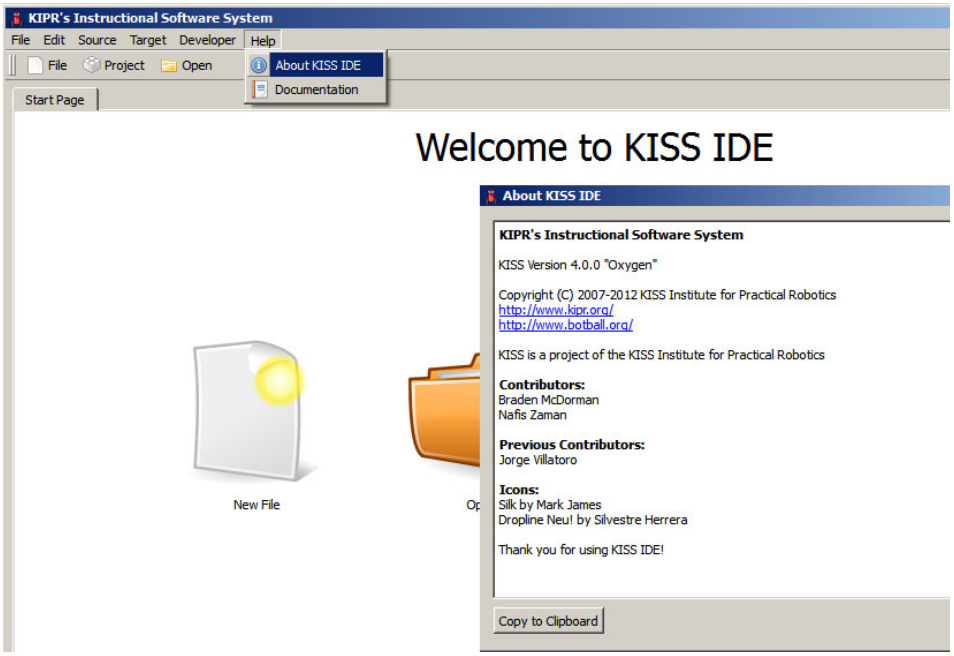
Botball Components

Camera

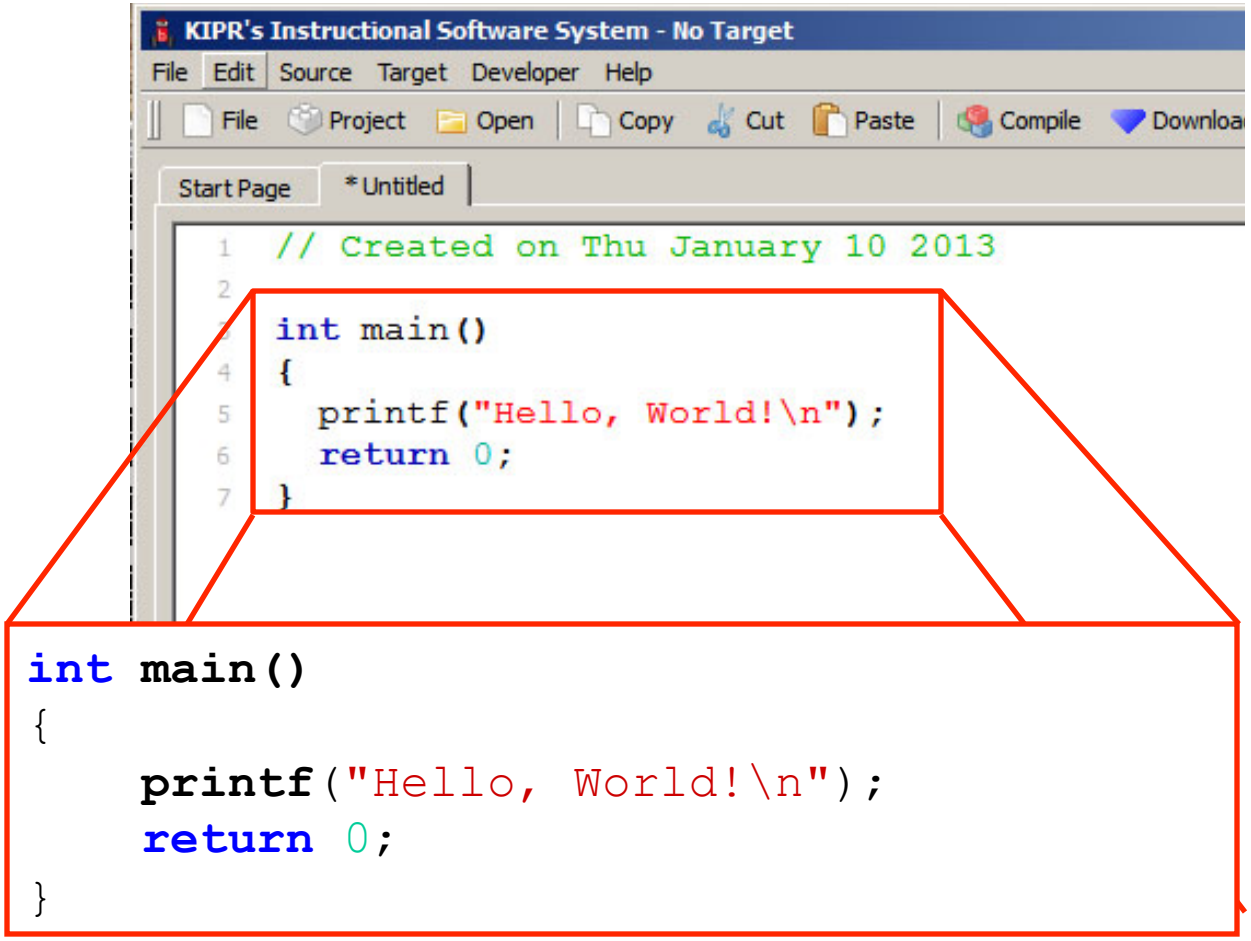


The KISS IDE

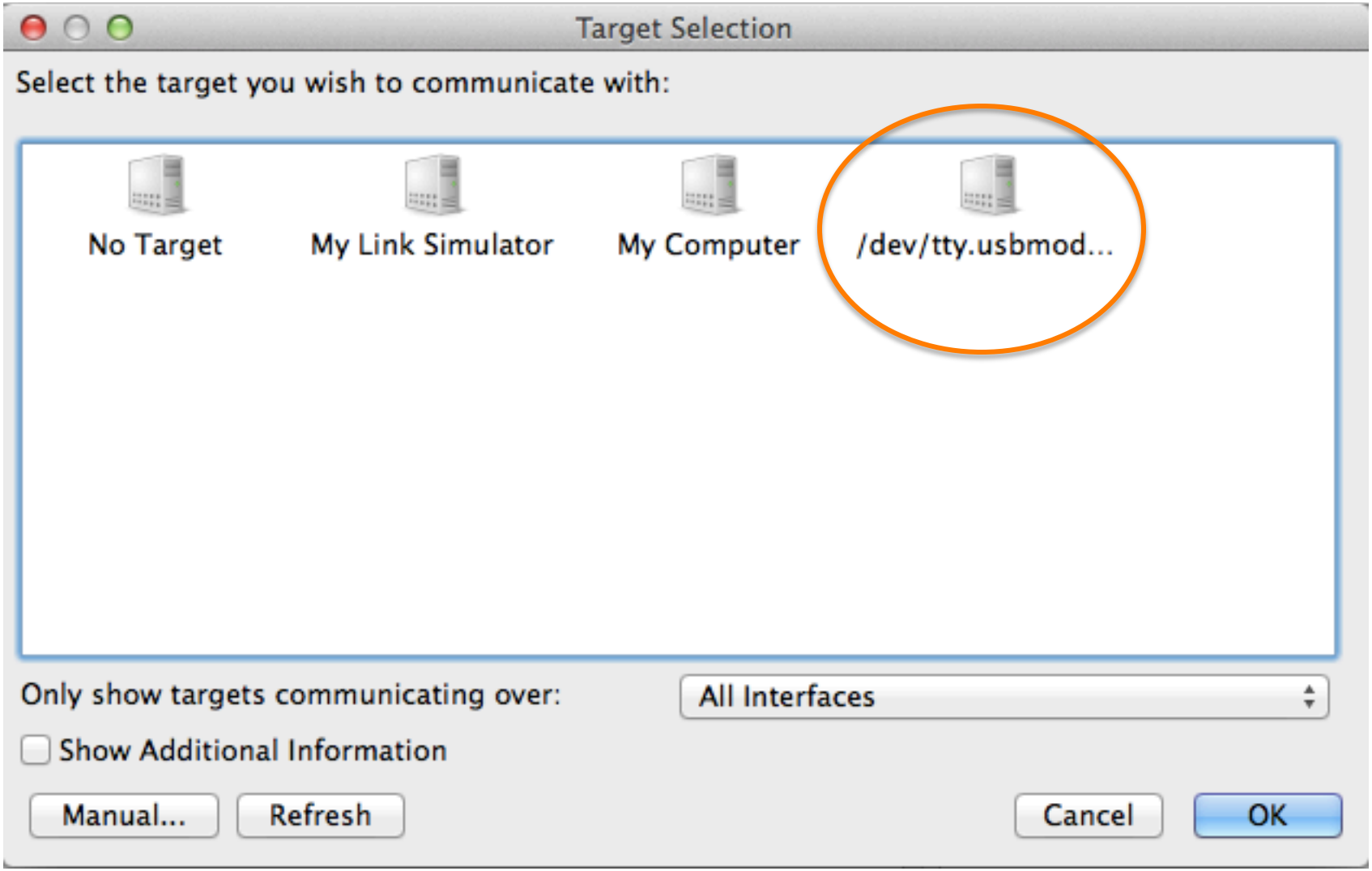
- http://www.kipr.org/software
  - Link controller
- C programming language



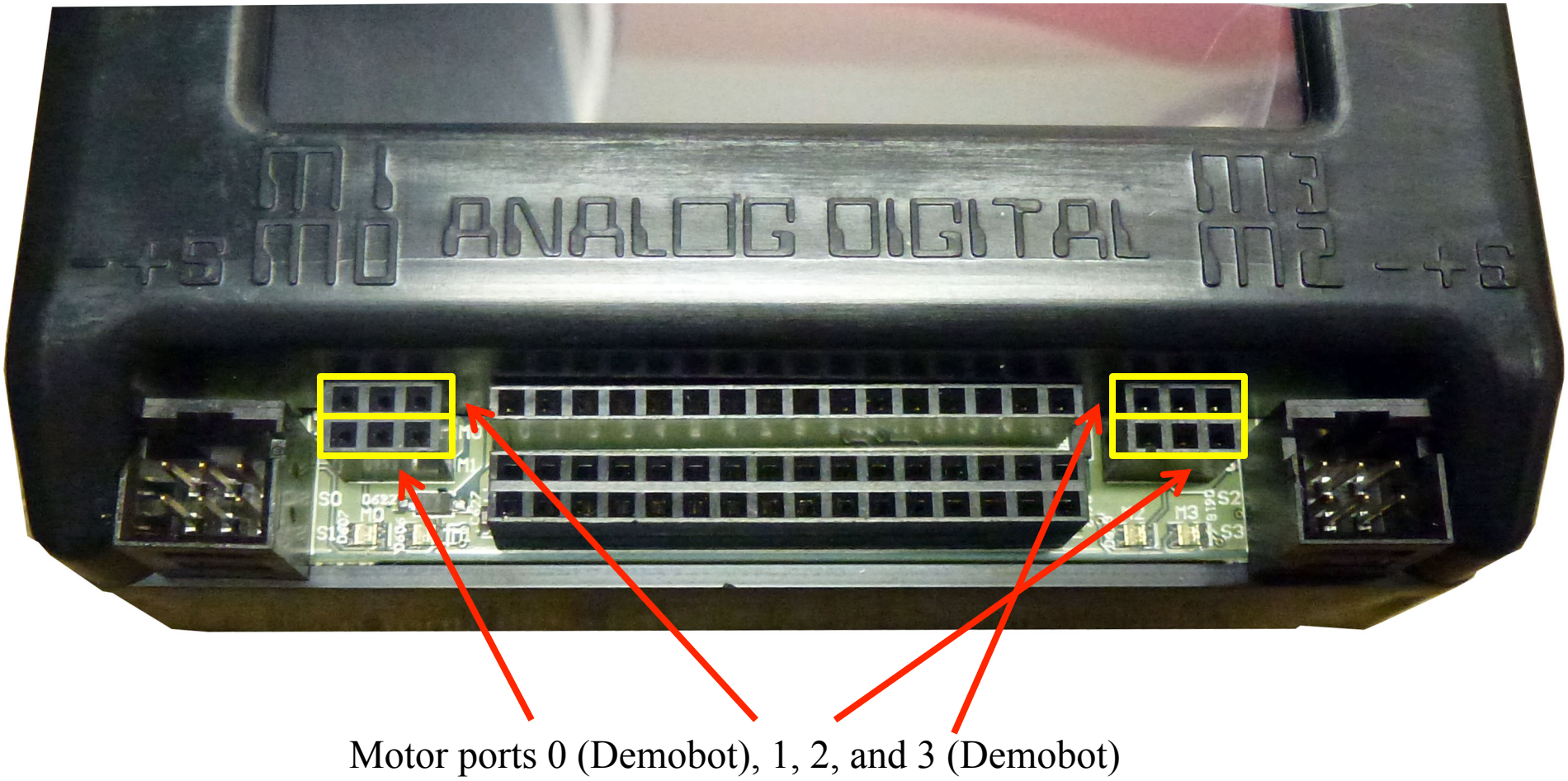
A Simple C Program



Connecting Link Controller

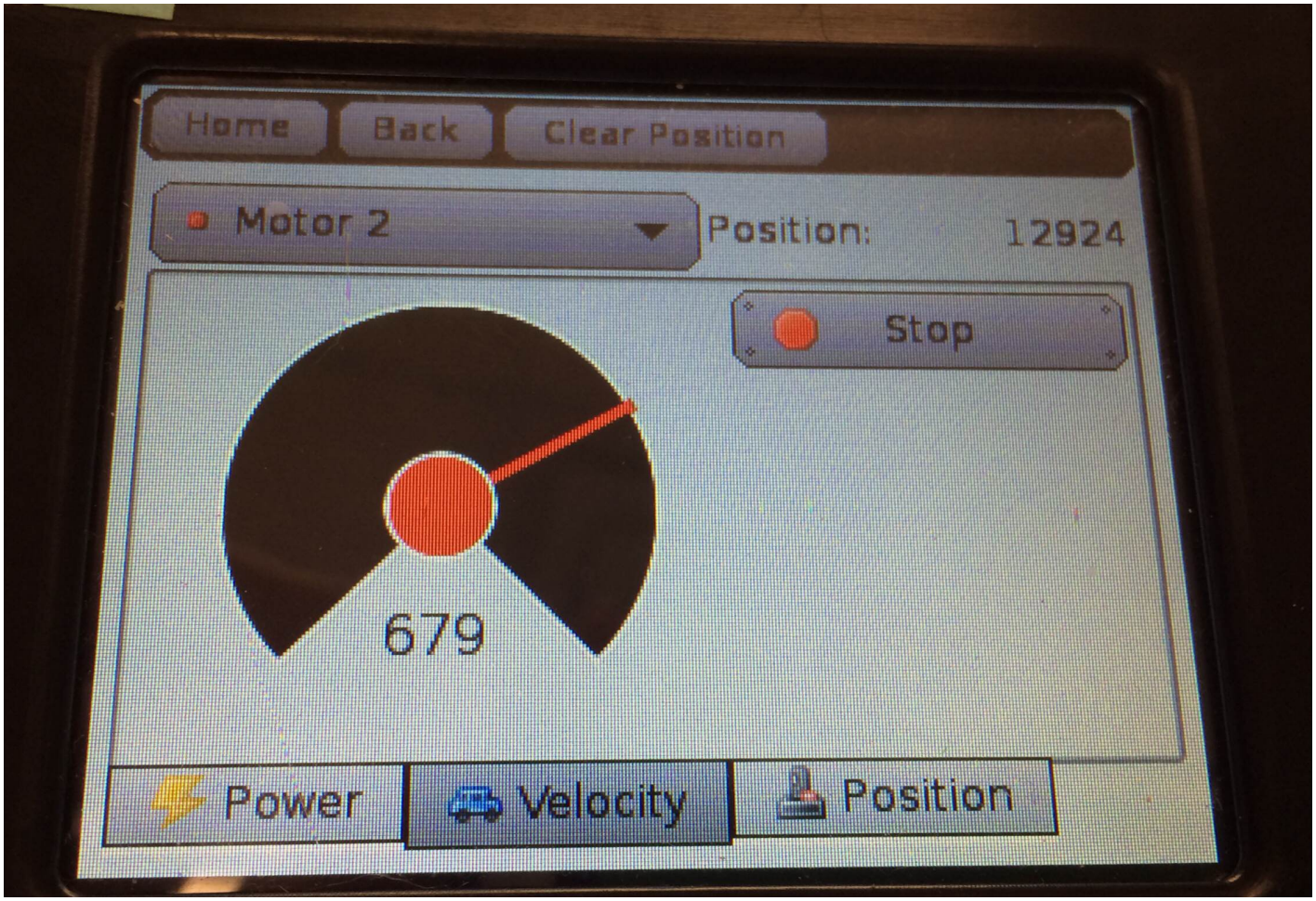


KIPR Link Motor Ports



Motor ports 0 (Demobot), 1, 2, and 3 (Demobot)

KIPR Link Motor: On-Screen Control



KIPR Link Motor Commands

`mav(), ao(), off()`

- **`mav(`***motor#`,`vel`);`** [mav=move at velocity]
  - *motor#* is the motor port (0-3) being used
  - *vel* is the rotational speed of the motor measured in ticks per second (-1000 to 1000)
  - the amount of rotation per tick depends on the kind of motor
  - the motor runs at the set speed until turned off or commanded otherwise
- **`ao();`** turns off all motor ports
- **`off(`***motor#`);`** turns off the specified motor port

Motor Position Counter

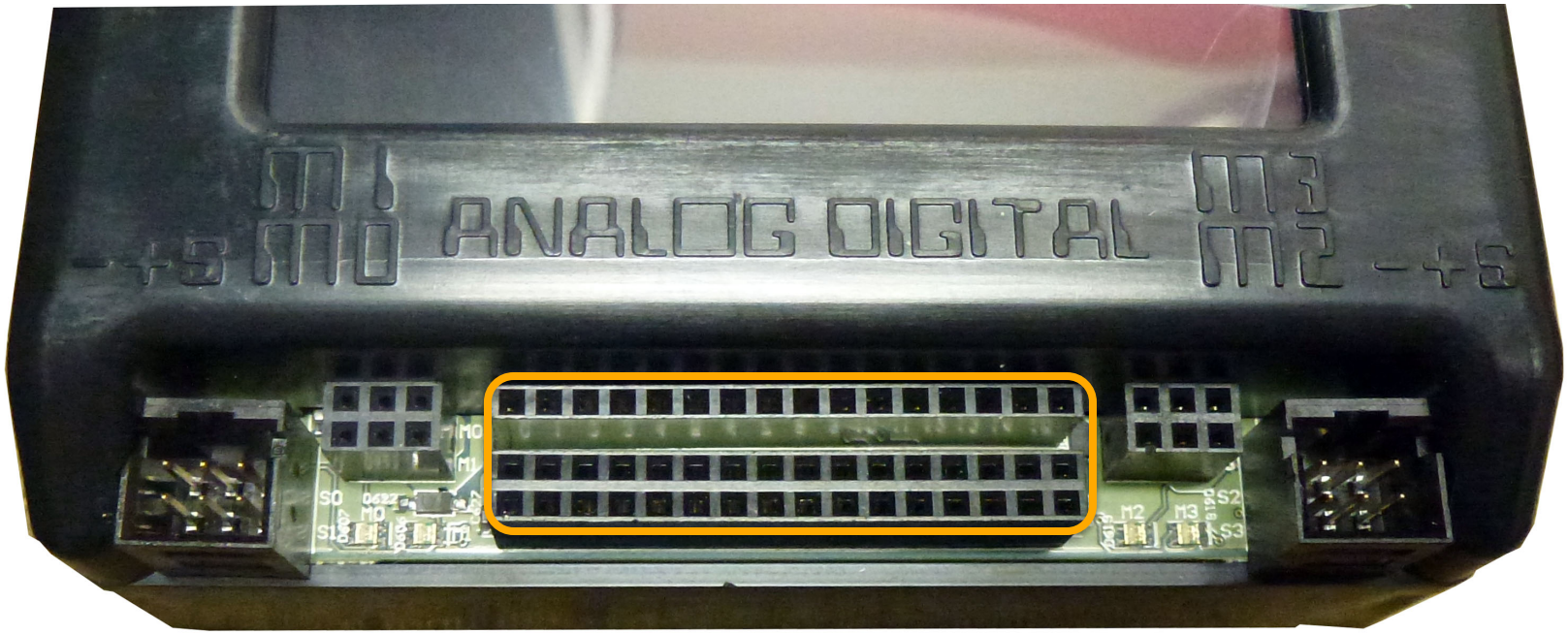
- As a DC motor runs, the KIPR Link keeps track of its current position in ticks
  - **`get_motor_position_counter(`***motor#***`);`**
    is a library function that returns this value for *motor#*
  - **`clear_motor_position_counter(`***motor#***`);`**
    is a library function that resets the *motor#* counter to 0
  - You can see the current value of the counter for a motor on the *motors..test* and *Sensor Ports* screens

Ex.1 Motor Control

```
3   int main()
4   {
5       // waiting for the A button to be pressed to start
6       while(!a_button())
7       {
8       }
9
10      // go straight
11      printf("go straight....\n");
12      mav(2, 300);
13      mav(0, 300);
14      msleep(3000);
15      printf("stop motor...\n");
16      mav(0, 0); // all motors stop
17      mav(2, 0);
18      msleep(2000);
19      // turn right
20      printf("clear counter\n");
21      clear_motor_position_counter(0);
22      int position = 0;
23      mav(0, 180);
24      mav(2, -180);
25      while(position < 700) {
26          position = get_motor_position_counter(0);
27          printf("position = %d\n", position);
28          msleep(500);
29      }
30      off(0);
31
32      return 0;
33  }
```

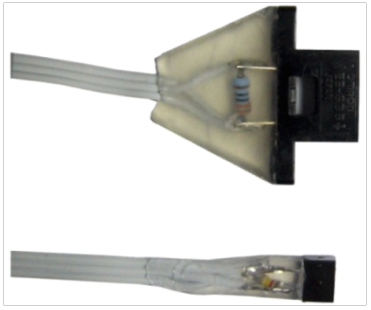
KIPR Link Sensor

- Plug the reflectance sensor into analog ports on the KIPR Link
- If you aim the reflectance sensor at the table and move it across the table edge its value will change



IR Reflectance Sensors

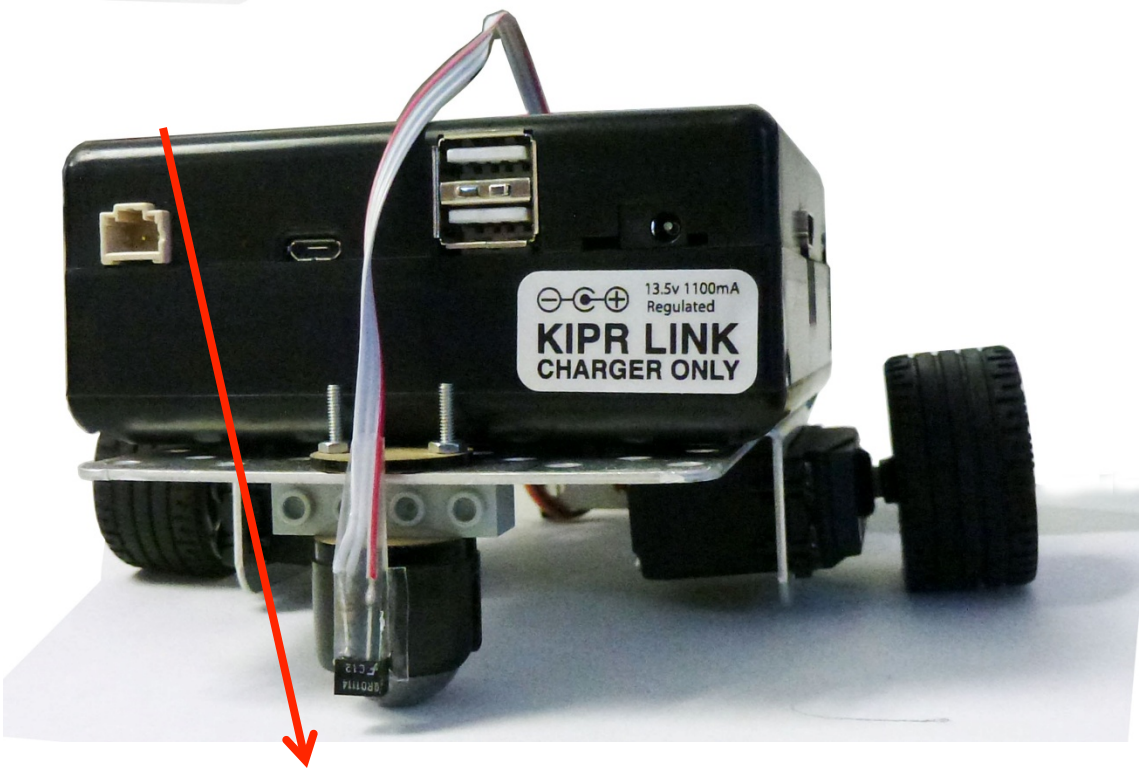
- An IR reflectance sensor has an emitter producing an IR beam and an IR light sensor that measures the amount of IR reflected when the beam is directed at a surface
- There are two reflectance sensors in the Botball kit



- The KIPR Link library function **`analog10`** returns an amount that measures the amount of light reflected as a value between 0 and 1023
- A dark spot reflects less IR, producing a higher reading

Line Following

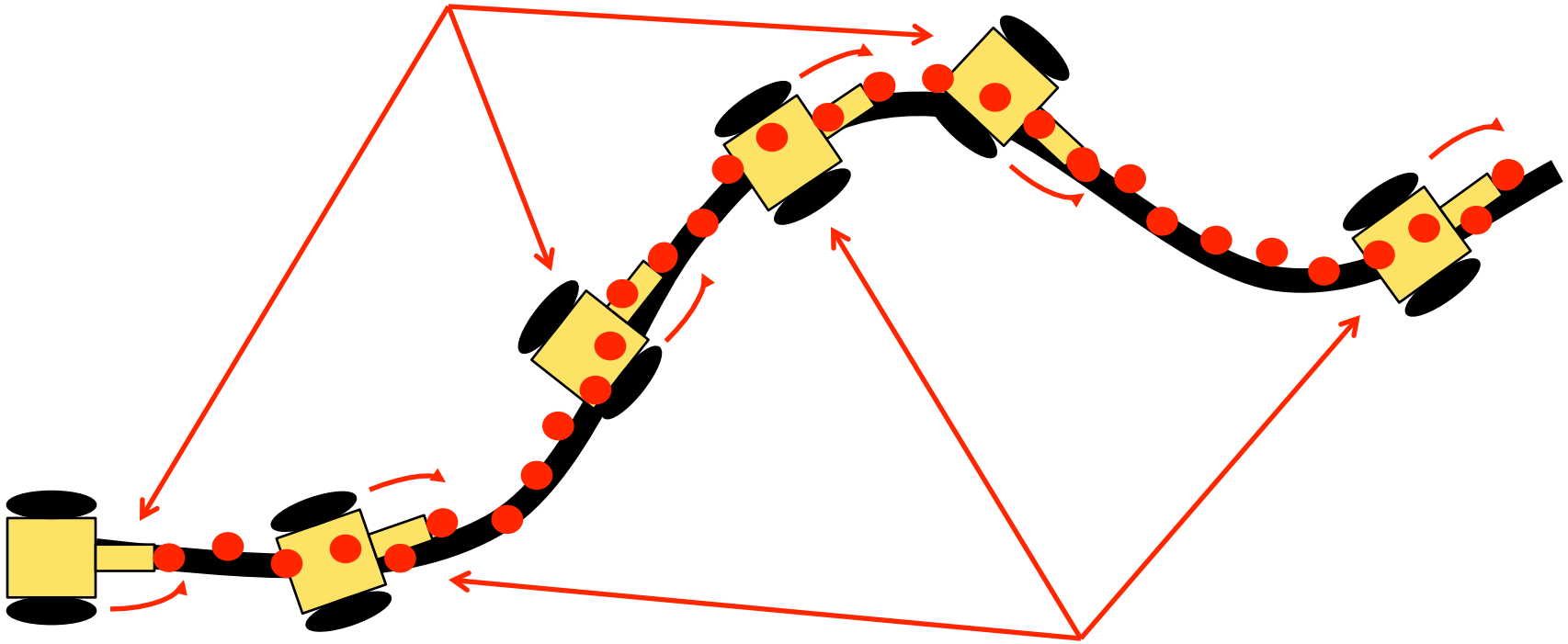
- Have a robot follow a line it can detect using a reflectance sensor



Line Following Strategy

Follow the line's left edge by alternating the following 2 actions:

1. If detecting dark, arc left



2. If detecting light, arc right

Line Following

```
/* Line following with a single sensor: arc left when the
    reflectance sensor detects dark and otherwise arc right
 Use the Sensor Ports screen to find the high & low
    reflectance sensor values for the robot on the surface
 Set the threshold halfway between                          */
int main()
{
  int rport=7, leftmtr=0, rghtmtr=3;  // identify port and motors
  int threshold=512;          // set threshold for light conditions
  int high=100,low=-10;       // set wheel powers for arc radius
  printf("Line following: position robot on tape\n");
  printf("Press B button when ready\n\nPress side button to stop\n");
  while(b_button()==0) {}     // wait for button press
  while(side_button()==0){              // stop if button is pressed
     while (analog10(rport) > threshold) { // continue until not dark
       motor(leftmtr,low); motor(rghtmtr,high);   // arc left
       if (side_button()!=0) break; }     // or button pressed
     while (analog10(rport) <= threshold){     // continue until dark
        motor(leftmtr,high); motor(rghtmtr,low);  // arc right
       if (side_button()!=0) break; }     // or button pressed
  }
  ao();  // stop because button pressed
  printf("done\n");
  return 0;
}
```

Servo Motors

- A servo is designed to rotate to a specified position and hold it. To save power, servo ports by default are not active until enabled.
- A command is provided in the KIPR Link library for enabling (or disabling) all servo ports
  - `enable_servos();` activates all servo ports
  - `disable_servos();` de-activates all servo ports

- `set_servo_position(2,925);` rotates servo 2 to position 925
  - Default position when servos are first enabled is 1024

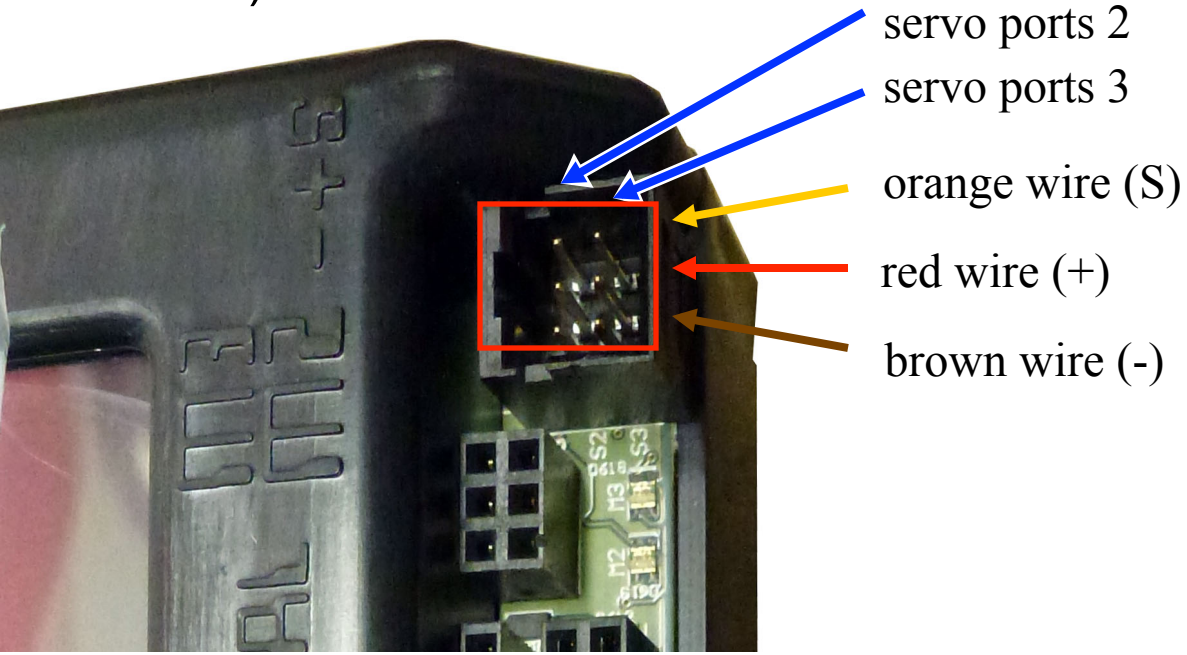
- `get_servo_position(2);` provides the current position of servo 2
  - Works only when servos are enabled

KIPR Link Servo Motor Ports

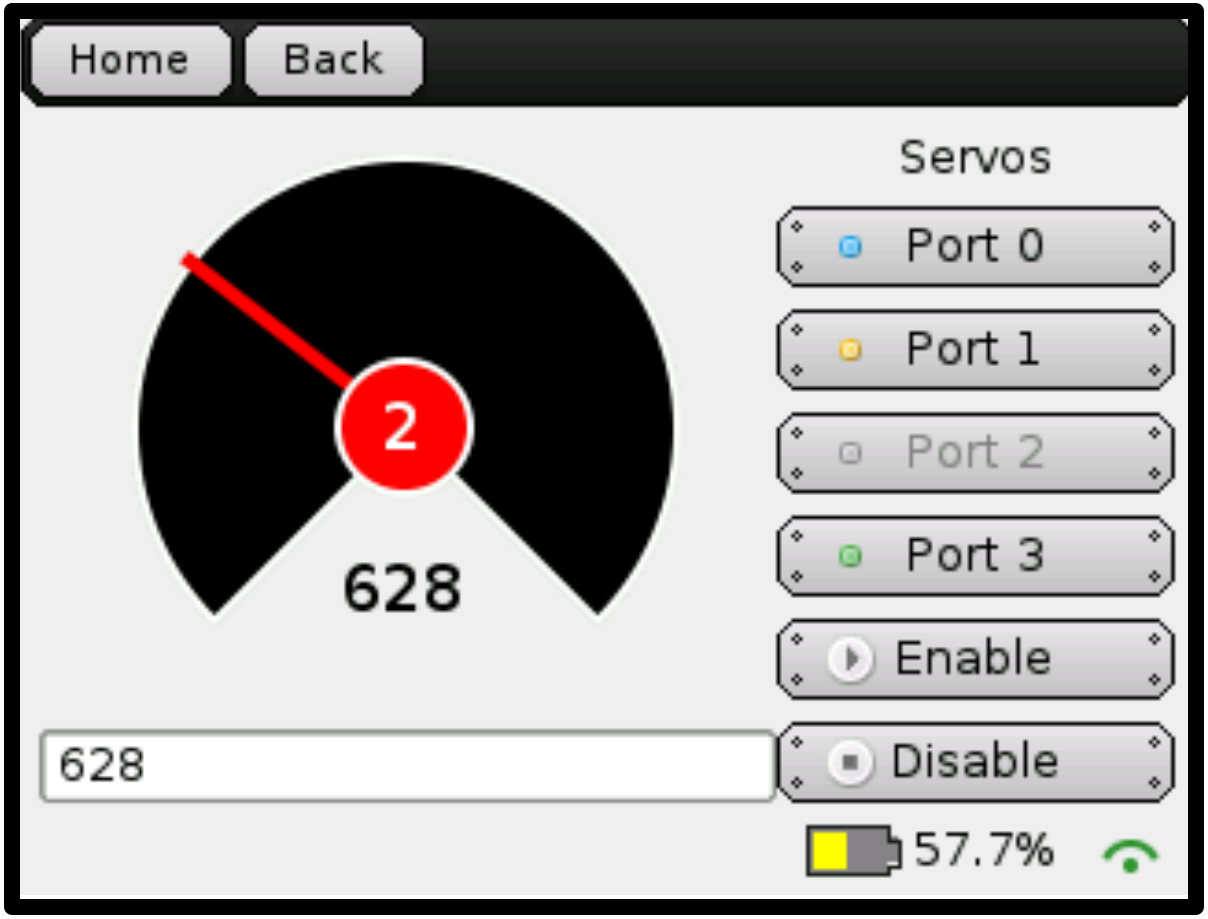
servo ports 0 and 1; servo ports 2 and 3

Plugging in Servos

- The KIPR Link has 4 servo ports numbered 0 & 1 on the left and 2 & 3 on the right
- Plug orientation order is, left to right, brown-red-orange when the KIPR Link is oriented so the screen can be read (or follow the labeling: - + S; the orange signal wire goes in S)



KIPR Link *Servo* Screen

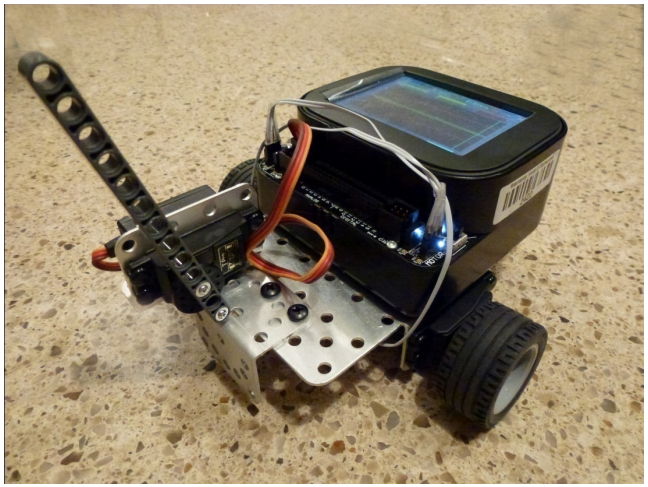


The KIPR Link *Servo Test* screen can be used to center a servo or determine what position values to use once the servo is installed on a bot

`while` Loop Operating a Servo

- Suppose we want to have a servo move from position 200 to position 1800 in steps of 100
  - with less effort, this can be done by using a `while` loop

```
set_servo_position(1,200); // move servo 1 to position 200
msleep(100); // give servo time to move
 while (get_servo_position(1) < 1800)
{ // move servo 1 in steps of 100
  set_servo_position(1,get_servo_position(1)+100);
  msleep(100); // give it time to move
}
```



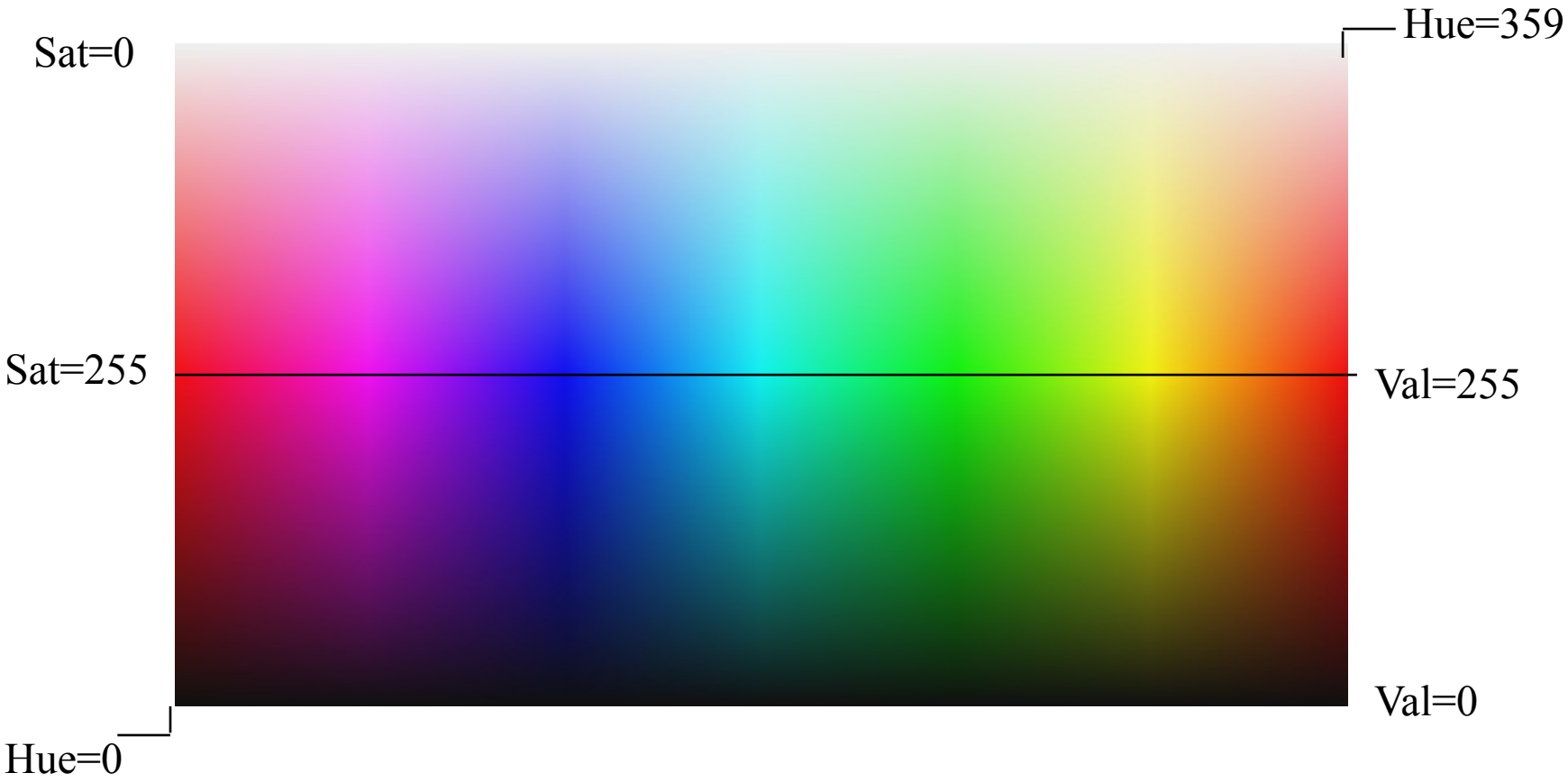
Vision: Camera

- Vision setup
- HSV color selection and color blobs
- Training the Link to use an HSV color model
- Library functions for using the camera

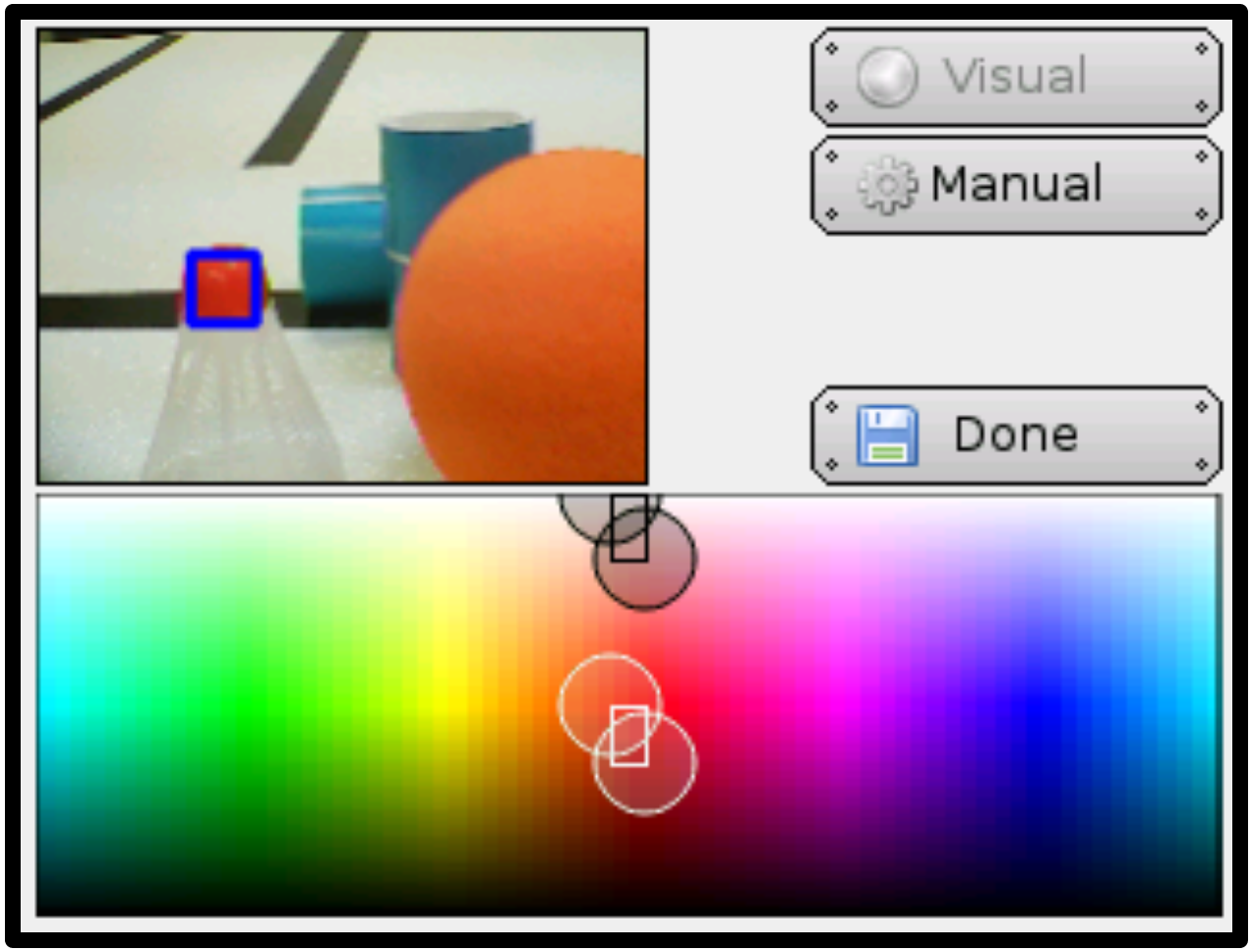
camera_open(<res>**)**
camera_close()
camera_update()
get_object_count(<ch>**)**
get_object_bbox(<ch>**,**<obj>**)**

get_object_center(<ch>**,**<obj>**)**
get_object_data(<ch>**,**<obj>**)**
camera_load_config(<name>**.conf**
)

HSV Color Selection Plane



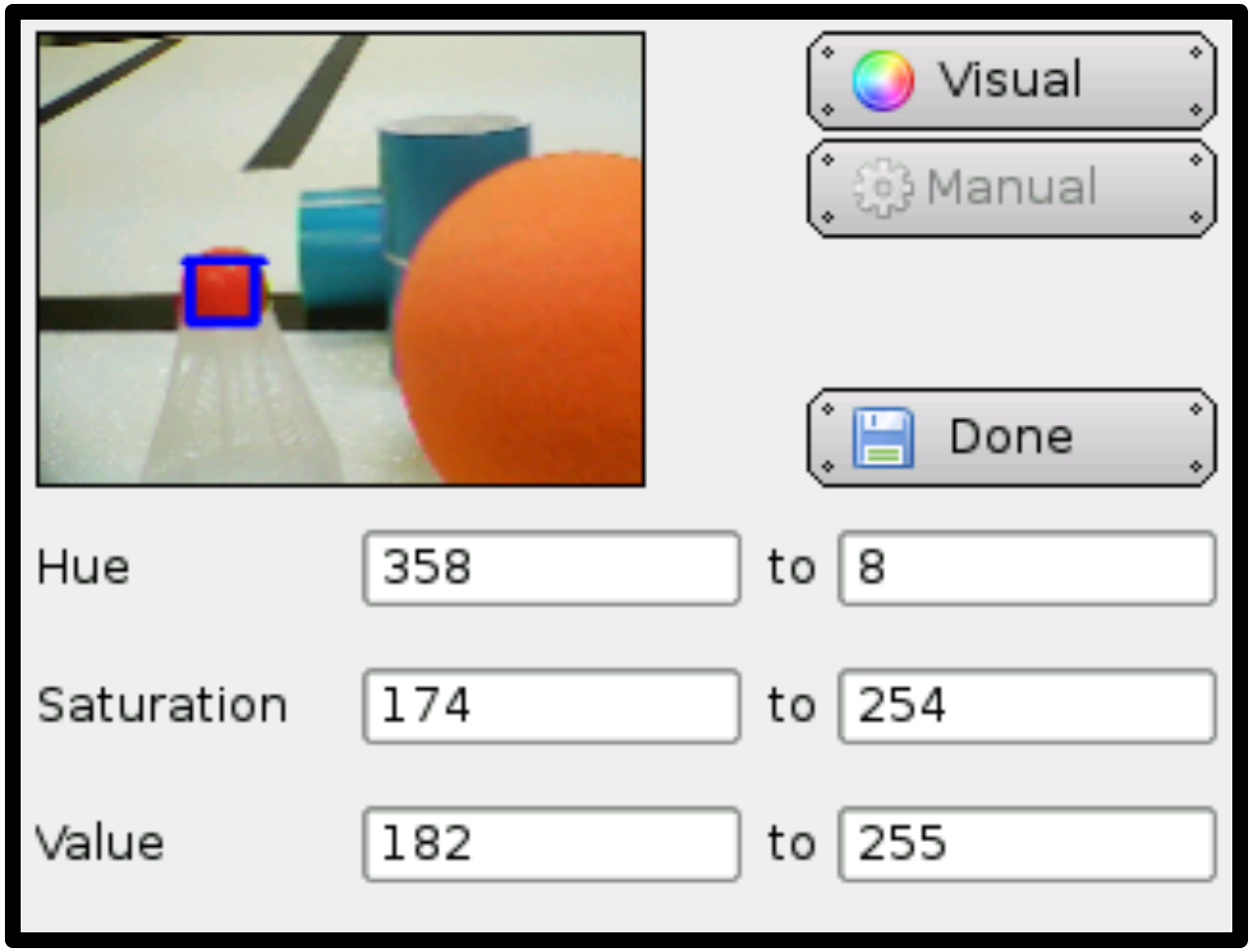
Channels Interface



Color Blobs

- Each pixel on the screen has an HSV color
- When we say "red", we really mean a range of HSV colors on the color selection plane that are approximately red
- Two rectangular pieces of the color selection plane that correspond to being "red" specify the range of HSV colors to be viewed as red by the KIPR Link
  - This is called an HSV color model
- A red *blob* is all contiguous pixels matching one of the HSV colors in the red range
- A blob has a bounding box, a center, etc.

Manual Channel Interface



Ex.2: Camera

```
4   #define numChannels 4
5
6   int main()
7   {
8      camera_open(LOW_RES);
9      while(side_button() == 0) {
10        camera_update();
11        printf("number of objects = %d", get_object_count(0));
12        int colors[numChannels];
13        int i = 0;
14        for(i = 0; i<numChannels; i++) {
15           colors[i] = get_object_count(i);
16        }
17
18        int maxCount = 0;
19        int maxIndex = 0;
20        for(i = 0; i < numChannels; i++) {
21           if(colors[i] > maxCount) {
22                 maxCount = colors[i];
23                 maxIndex = i;
24           }
25        }
27        switch(maxIndex) {
28        case 0:
29           printf("pink\n");
30           break;
31        case 1:
32           printf("purple\n");
33           break;
34        case 2:
35           printf("green\n");
36           break;
37        case 3:
38           printf("red\n");
39           break;
40        }
41
42        msleep(500);
43        printf("update**********\n");
44     }
45     return 0;
46  }
```

Ex. 3

```c
// Created on Mon September 16 2013
#define rmotor 0
#define lmotor 3
#define numChannels 4

int main()
{
  camera_open(LOW_RES);

  while(side_button() == 0) {
    //go forward
    mav(rmotor, 300);
    mav(lmotor, 300);
    msleep(1500);
    //stop
    ao();


    printf("sensing.....\n");
    camera_update();
    camera_update();
    camera_update();

    int colors[numChannels];
    int i = 0;
    for(i = 0; i<numChannels; i++) {
      colors[i] = get_object_count(i);
    }

int maxCount = 0;
int maxIndex = 0;
for(i = 0; i < numChannels; i++) {
  if(colors[i] > maxCount) {
      maxCount = colors[i];
      maxIndex = i;
  }
}

switch(maxIndex) {
case 0:
  printf("pink\n");
  // turn right
  mav(rmotor, -300);
  mav(lmotor, 300);
  msleep(2300);
  break;
case 1:
  printf("purple\n");
  // go back
  mav(rmotor, -300);
  mav(lmotor, -300);
  msleep(2300);
  break;
case 2:
  printf("green\n");
  // turn left
  mav(rmotor, 300);
  mav(lmotor, -300);
  msleep(2300);
  break;
case 3:
  printf("red\n");
  // turn 180
  mav(rmotor, 300);
  mav(lmotor, -300);
  msleep(4600);
  // go forward
  mav(rmotor, 300);
  mav(lmotor, 300);
  msleep(1000);
  break;
}


  // stop
  ao();
  msleep(500);
  //printf("update***********\n");

} // end while

  ao();
  printf("done\n");

  return 0;
}
```