CAP4630 Artificial Intelligence Robotics using Botball

Dr. Ching-Hua Chuan

Outline

- A few things to keep in mind
  - Charging
  - Manuals and sites
  - Hardware
- Activities in the Botball workshops
- AI Botball challenges

Charging the KIPR Link Controller

- For charging the KIPR Link, **use only the power supply which came with your Link**
  - **Damage to the Link from using the wrong charger is easily detected and will void your warranty!**

- The KIPR Link power pack is a lithium polymer battery so the rules for charging a lithium battery for any electronic device apply
  - You should NOT leave the unit unattended while charging
  - Charge away from any flammable materials and in a cool, open area

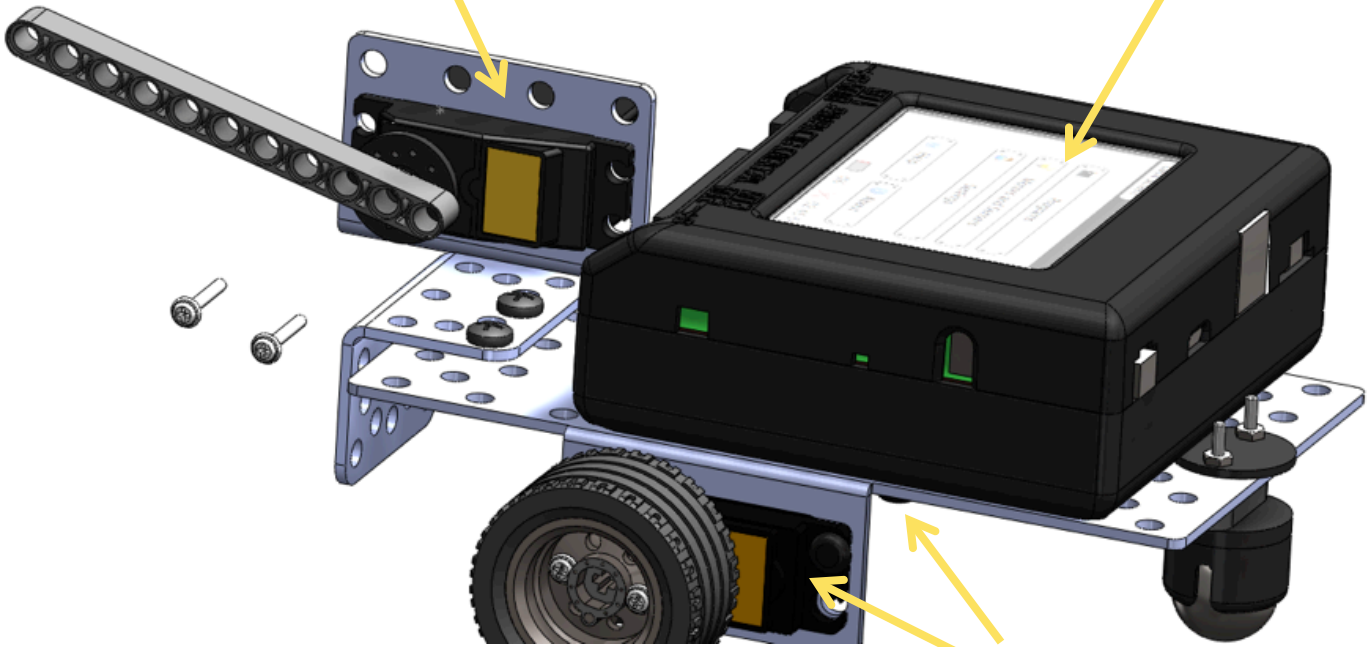
Building The Bot

Long level sensor
(digital)

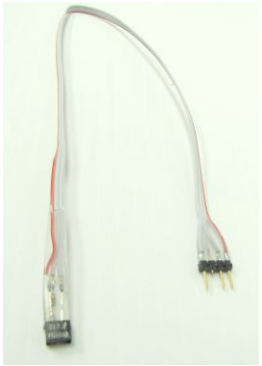
USB camera

servor

LINK controller



motors



IR sensor
(analog)

Useful Documents and Sites

- KISS Institute for Practical Robotics (KIPR)
  - http://www.kipr.org/hardware-software (IDE)
- LINK controller getting started manual
  - http://files.kipr.org/link/documentation/KIPR_Link_Manual_2015.1.1.pdf
- Sensor & motor manual
  - http://www.cs.uml.edu/~holly/teaching/91450/spring2015/Sensor_and_Motor_Manual_BB2011.pdf
- Robot building guide
  http://files.kipr.org/link/documentation/2015_Robot_Building_Guide.pdf
- Botball community site
  - http://community.botball.org/

Important Components in Botball

- Technical skills
  - STEM and AI
  - Documentation
- Soft skills
  - Project management
  - Teamwork
  - Communication

Hardware: Motor

- Motors: take electrical energy and convert that to rotational mechanical energy.
- Two wheel direct drive:
  - Two motors, one on each side
  - A skid or castor on the front or back
  - Turning
    - Drive one motor forward and one backward
    - Drive both motors forward at different speed so that the robot can drive in an arc.

SG-5010 Continuous Rotation Motor

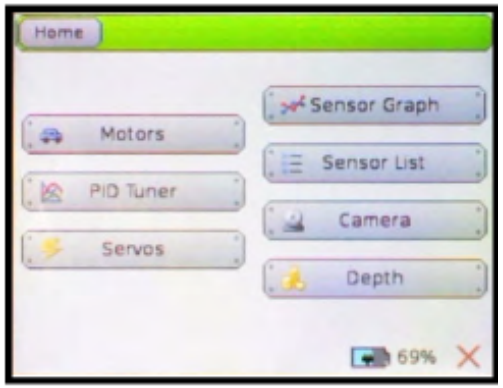
Performance

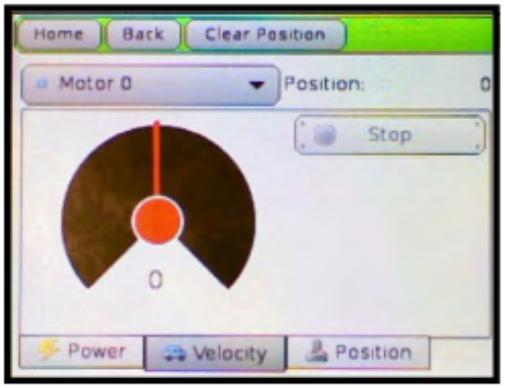
Torque: 156oz in
Speed @ 60°: 0.11 sec

CS-65 LEGO Motor

Testing The Motors

- Built in motor test





- Checking motor polarity
  - The LEDs in front of the motor port: blue for moving forward and red for moving backwards.

Hardware: Motor

- Speed and power
  - Turn on a motor at a scaled pulse width modulation (PWM) percentage
  - Power levels range from 100 (full forward) to -100 (full backward)

- Ticks
  - A tick is the smallest measurable amount the motor can turn.
  - The controller uses ticks to measure the position of the motor.
  - As each motor operates, the Link controller keeps updates a position counter for the motor to keep track of how far it has rotated.
  - Typically 1100 ticks per full rotation.

KIPR Link Library for DC Motors

```
void clear_motor_position_counter(int m);
int get_motor_position_counter(int m);
void motor(int m,int p);
void mav(int m,int vel);
void mrp(int m,int vel,int ticks);
void bmd(int m);
void ao();
void off(int m);
```

Work better than mav, especially in high speed

m: motor #, {0, 1, 2, 3}
p: power, [-100 100]
vel: velocity (in ticks), [-1000 1000] for mav and [0 1000] for mrp

Hardware: Servo

- A servo (typically used in an arm) motor's function is to move to a position and hold it (the motor will continue to draw significant power to maintain position).
- A servo will try to get to the position it is set, even if it means straining or breaking the servo.
- Make sure not to exceed the max torque limit of the servo.

SG-5010 Standard Servo

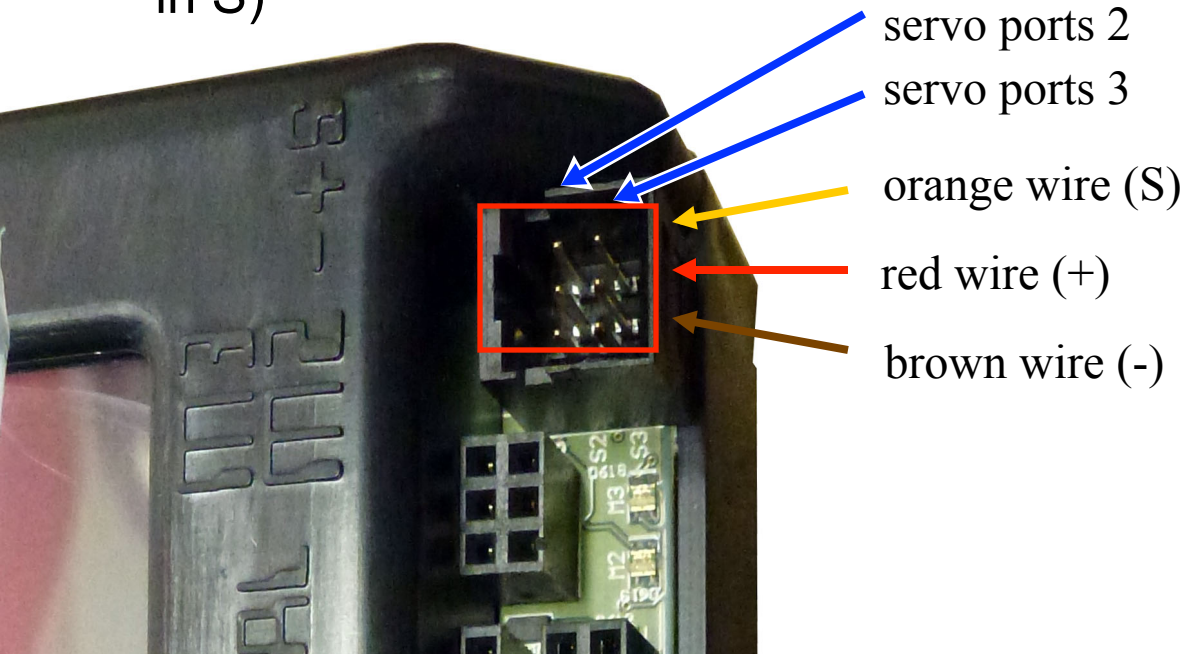
Performance

Torque: 156oz in
Speed @ 60°: 0.11 sec

CS-60 Standard Servo

Plugging in Servors

- Servo motors (brown/black-red-yellow cables with 3 prong receptacle) plug into the KIPR Link servo ports
- 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 Library for Servos

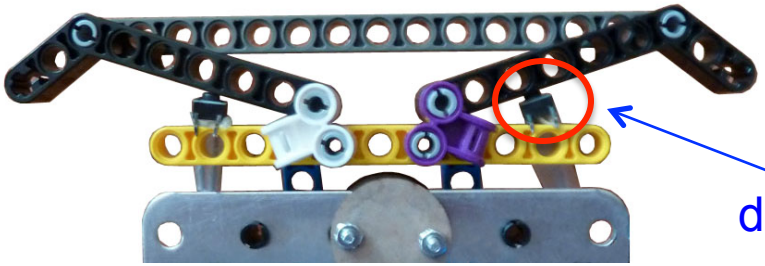
```
void enable_servo(int servo);
void disable_servo(int servo);

// set a position for the servo port
set_servo_position(2, 670);
```

- Servo position ranges from 0 to 2047.
- Call enable_servo() without a position will set the servo to the middle position (1024).
- Call disable_servo() to disable power for a specific servo can save battery when the servo is not needed.

Activity: Building A Bumper

- Digital on/off sensors
  - the function `digital(<port>)` returns 0 if that bumper is not pressed and 1 if it is pressed
- Build a two bumper sensor and make sure the sensors can activate independently
- If the robot runs into a wall with the bumper it rotates until both sensors are activated (go backward)



digital sensors

An Example: Bumper

```
/*****************************************************************
Drive the simulated robot forward at half power till it bumps
*****************************************************************/
1. int main()
2. {
3.   printf("Drive Straight till bump\n"); // announce the program
4.   msleep(1000); // wait 1 second
5.
6.   while (digital(9)==0 && digital(8)==0) // check bumpers
7.   {
8.      motor(0,55); // Drive left wheel forward at 55% power
9.      motor(2,50); // Drive right motor forward at half power
10.   }
11.
12.  ao(); // Stop motors
13.  printf("All Done\n"); // Tell user program has finished
14.  return 0;
15. }
```

Tip: Time and Sensing

- If your robot uses msleep to drive for a specified time, it is literally "sleep moving" and will not be monitoring bumpers, buttons or other sensors
- The function seconds() returns a value of type double that represents the Link's internal clock.
- By getting the difference between the current value of seconds() and one stored from an earlier time, you can get the elapsed time.

```
while((seconds()-start)<10.0 && //check the time & bumps
        digital(9)==0 && digital(8)==0){
  motor(0,55); motor(2,50);
} //exit loop when time is up or bumpers are bumped
```

Tip: Timing for Botball

- When executed, the function

  shut_down_in(<*game_secs*>**);**

  starts a process that turns off all motors after *game_secs* has elapsed and keeps any new commands from being processed

```
int main()
{
   printf("Program stops in 3 sec\n");
   shut_down_in(3.0);
   while (side_button()==0)
   {   beep();  msleep(200);      }
   printf("All done\n");  // shuts down before this!
   return 0;
}
```

Functions in C Program

```
/******* If A turn left, if C turn right (mirror behavior) *******/
1. void turn_left(double seconds);  // prototype for turn_left
2. void turn_right(double seconds); // prototype for turn_right


3. int main() {
4.   printf("Side button to stop\n"); // announce
5.  // Loop until the side button is pressed.
6.   while (side_button() == 0) {
7.     // If the 'A' button is pressed, then turn left
8.     if (a_button() == 1) { turn_left(1.0)); }
9.     // Else if the 'C' button is pressed, then turn right
10.    else if (c_button() == 1) { turn_right(1.0); }
11.  }
12.  printf("All Done\n"); // Tell user program has finished
13.  return 0;
14. }

15. /*Function definitions go below*/
16. void turn_left(double seconds) {
17.     motor(0, 50); motor(2, -55); msleep(seconds);
18. }
```

Activity: Controlling Servors

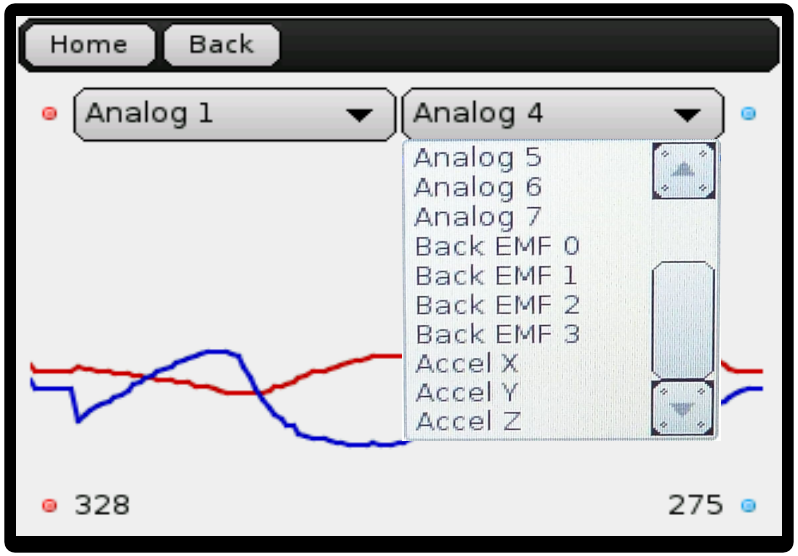
Before

After

- Have the robot detects when it is tilted, then stops the servo motion
- You should rely on the accelerometer values, not the servo position

Accelerometer

- An accelerometer measures force accelerating an object in 3 directions (vertical z, horizontal x, and horizontal y)
  - The y direction is front-to-back on the KIPR Link, x is left-to-right
  - Midpoint of 0, -512 to 511 range
- For an object at rest or moving on a flat surface at a constant speed the accelerometer measures no force for x and y
  - Gravity always exerts a force, so z > 0



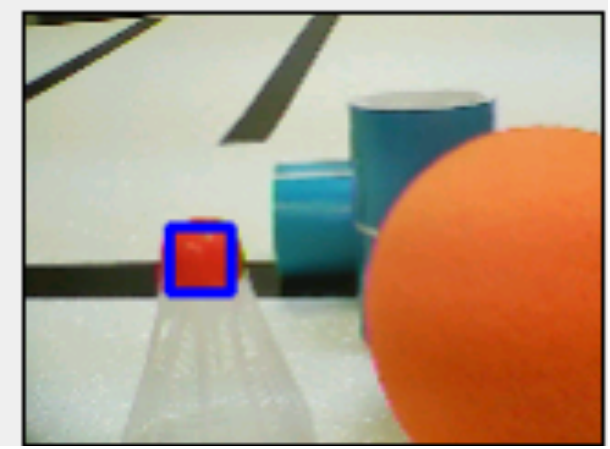
Activity: Controlling Servors

```
/********  Stop when accelerometer shows robot has tilted
**************************************************************/
1. int main() {
2.     // preset servo 1 position
3.    printf("advance using A button\n\nB to quit\n");
4.    set_servo_position(1,200);
5.    enable_servos(); // turn on servos
6.    msleep(2000); // pause while it moves and user reads screen
7.    while((accel_y() > -150) && (b_button()==0))
8.    {  // move servo 1 in steps of 100
9.       set_servo_position(1,get_servo_position(1)+100);
10.      printf("servo at %d\n", get_servo_position(1));
11.      msleep(200); // pause before next move
12.      while((!a_button()) && (!b_button())) {}
13.   }
14.   disable_servos();
15.   printf("Tilt! Robot is done\n");
16.   return 0;
17. }
```

Vision

- Video of setting color models
- **http://youtu.be/nSszFa7opMA**

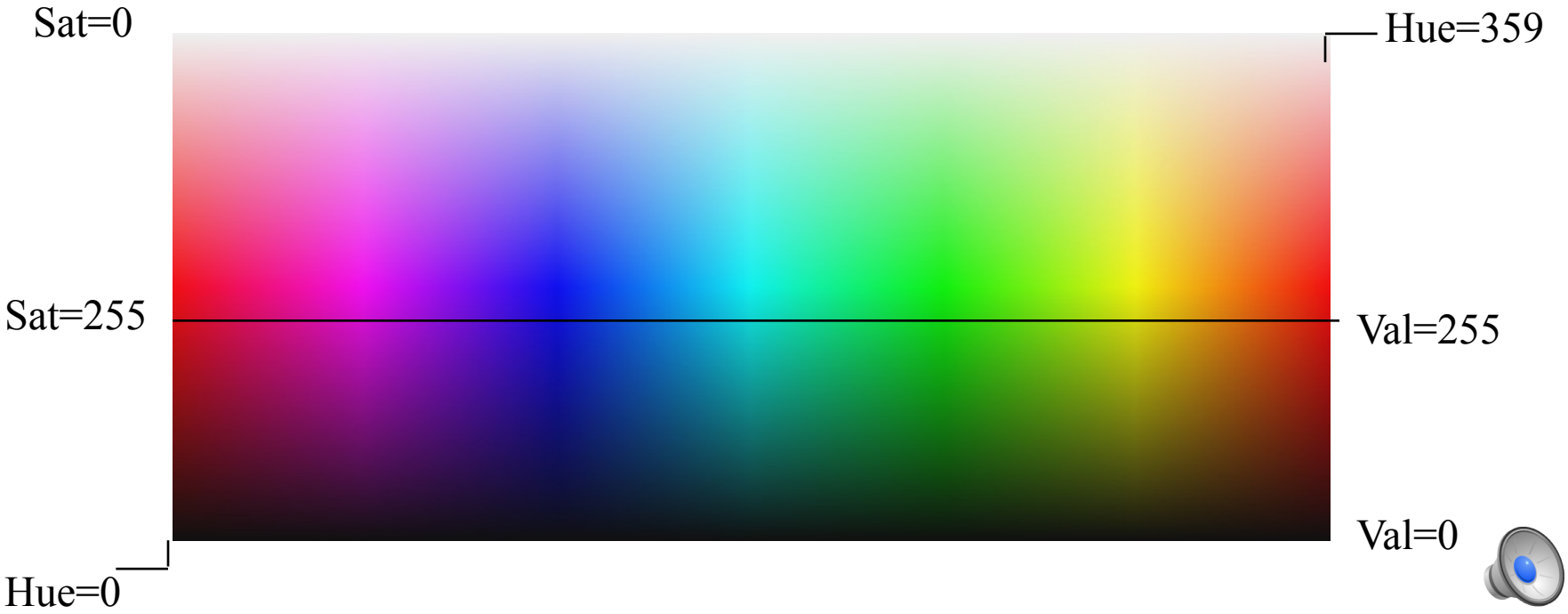
- You can create multiple vision system configurations
  - Each configuration can have up to 4 channels
- **YOU MUST SET ONE CONFIGURATION AS THE DEFAULT**
- The KIPR Link can handle 4 Channels simultaneously



USB camera

Color Blobs

- Each pixel on the screen has an HSV color
- 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.



Vision Performance Factors

- Focus
  - A slightly blurred image smooths out colors and can improve some tracking reliability
  - Adjust focus by turning the focus ring on the camera
- Image resolution
  - The lower the resolution the higher the frame rate
  - This is set by the argument given to camera_open(<res>)
    - HIGH_RES sets the image to 640x480
    - MED_RES sets the image to 320x240
    - LOW_RES sets the image to 160x120 - recommended

Vision System Library Functions

- `camera_update()` is a command that causes the KIPR Link to capture the most recent camera frame

- `get_object_count(3)` provides how many objects are being seen by channel 3 **in the default configuration**

- `get_object_center(3,0).x` for channel 3, object 0, returns the value of the center x coordinate of the largest object

More Object Functions

```
get_object_center(<ch>,<obj>).x
get_object_center(<ch>,<obj>).y

get_object_bbox(<ch>,<obj>).ulx
get_object_bbox(<ch>,<obj>).uly

get_object_bbox(<ch>,<obj>).width
get_object_bbox(<ch>,<obj>).height

get_object_area(<ch>,<obj>)
```

An Example: Vision

```
1. int main() {          // Start up the camera and specify the resolution
2.   int x, y, color=0;  // set up for color channel 0 (red)
3.   camera_open(LOW_RES);
4.   printf("Looking for red\nPress A when ready\n\n");
5.   printf("Press B button to quit\n");
6.   while (a_button() == 0); // wait for A button
7.   while (b_button() == 0){ // run till B button is pressed
8.     camera_update(); // process the most recent image
9.     if (get_object_count(color) > 0){
10.      //get x, y for the biggest blob the channel sees
11.      x = get_object_center(color,0).x;
12.      y = get_object_center(color,0).y;
13.      printf("Biggest blob at (%i,%i)\n",x,y);
14.    }
15.    else{
16.      printf("No color match in Frame\n");
17.    }
18.    msleep(200);  // give user time to read
19.  }
20.  printf("Program is done.\n");
21.  return 0;
22. }
```

An Example: Vision Tracking

```
/* Move the robot towards the largest object on channel 0.
   Robots stops if no object is detected*/
1. int main(){
2.    int ch=0, leftmtr=0, rghtmtr=3;  // identify channel and motors
3.    int high=100,low=-10;       // set wheel powers for arc radius
4.    camera_open(LOW_RES);
5.    printf("Move towards object on channel 0\n");
6.    printf("Press B button when ready\n\nPress side button to stop\n");
7.    while(b_button()==0) {}     // wait for button press
8.    while(side_button()==0){    // stop if button is pressed
9.      if(get_object_count(ch)>0) { // if object is seen...
10.       if(get_object_center(ch,0).x < 65) {// if object is on left...
11.         motor(leftmtr,low); motor(rghtmtr,high);   // arc left
12.       }
13.       else { if(get_object_center(ch,0).x > 95) {// if object is on right...
14.           motor(rghtmtr,low); motor(leftmtr,high);   // arc right
15.         }
16.         else {motor(rghtmtr,high); motor(leftmtr,high);}//go straight
17.       }
18.     }
19.     else {ao();}
20.   }
21.   ao();  // stop because button pressed
22.   printf("done\n"); return 0;
23. }
```

Motion Control

- Bang-Bang control
  - is a control strategy that changes power to a new value without a transition such as first slowing down
- Proportional control
  - Speeding up then slowing down proportionally when it is getting close to the target
- Examples

AI Botball Challenge 1

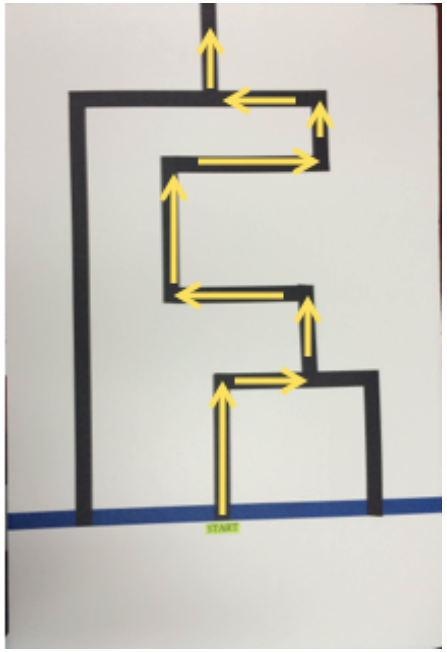
Goal:

- Walk between the lines as far and fast as it can.
- It fails once it stands on or crosses the line

Scores:

- 20 points for walking between the lines for 4 feet
- 15 points for walking for 3 feet
- 10 points for 2 feet
- 5 points for 1 feet
- 5 extra points for the team which reaches the farthest distance at the fastest speed

AI Botball Challenge 2



Goal:

- Get out of the maze

Scores:

- 20 points for getting through the maze
  - 5 extra points if the robot complete the maze within 1 minute
- 10 points for detouring back to the starting line
- 0 point otherwise

AI Botball Challenge 3

Push left Push right Push forward Pull back

Goal: recognize the color correctly and perform the corresponding action
Scores: 5 points for each correct movement