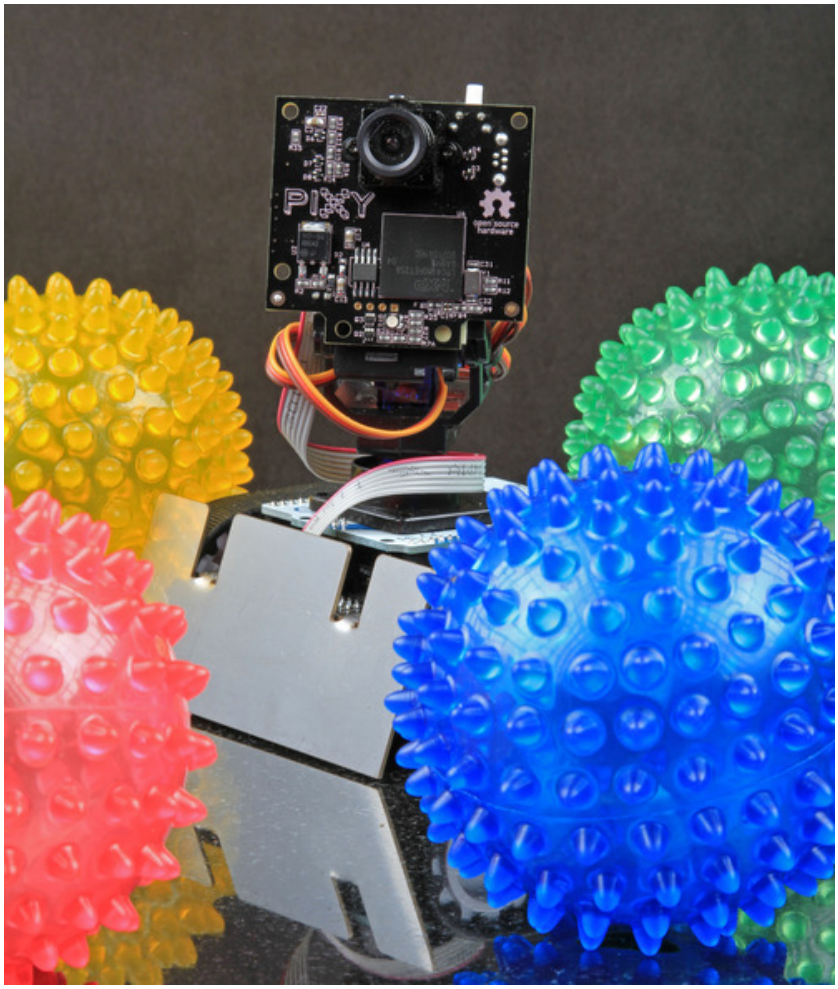
adafruit learning system

Pixy Pet Robot - Color vision follower

Created by Bill Earl



Last updated on 2015-06-19 12:30:09 PM EDT

Guide Contents

Guide Contents 2
Overview and Materials 4
Materials: 5
Tools: 5
Assemble the Camera 8
- Preparing the Pan/Tlt Base 8
- Remove the Side Tabs 9
- Trim the remaining tabs flush 10
- Cut a notch for the cable connectors 11
- Attach the Camera 13
- Connect the Servo Cables 15
- Secure the Cables 15

Final Assembly 17
- Attach the Camera to the Leonardo 17
- Connect the ribbon cable 18
- Attach the Camera and Processor to the Zumo 20

Playing with your Pixy Pet! 23
Teach the Camera 23
- Find a toy! 23
- Connect the Camera 23
- Run PixyMon 23

Upload the Code 24
Play Ball! 24
The Code 25
Pixy Pet Code Design 31
Tracking Objects 31
Following Objects 34
Feedback Control Basics 37
Measurements, Setpoints, Errors and Ouputs 37
Types of Control 37
- On/Off Control 37
- PID Control 38
- Proportional Control 38
- Integral Control 38

Derivative Control 38
Troubleshooting 40

Overview and Materials

This project pairs the super-awsome Pixy CMUCam-5 vision system with the high performance Zumo robot platform, a pan/tilt mechanism and an Arduino Leonardo for a brain.

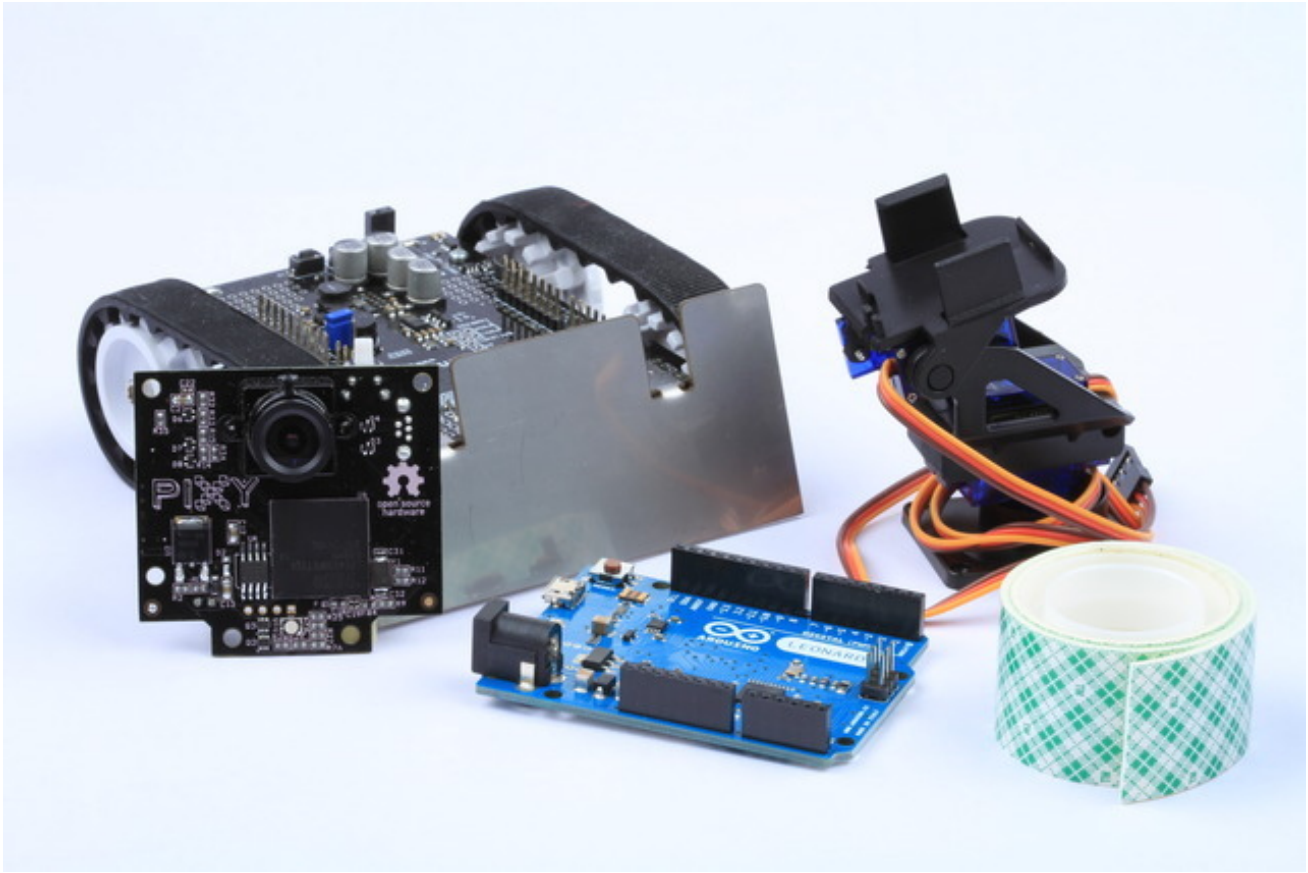
The Pixy camera has powerful image processing capabilities that can track objects by color. It can track dozens of objects simultaneously and report their locations to the Arduino in real-time. The built-in pan/tilt servo control is fast enough to track a bouncing ball.

The Zumo robot is a low-profile tracked robot platform designed for an Arduino controller. It uses two 75:1 precision micro metal gearmotors to drive extra grippy silicone rubber treads. Zumo has traction and torque to spare, with a top speed of approximately 2 feet per second (60 cm/s). This makes it a nimble little bot that can zip along at high speed and still turn on a dime.

Putting all this together with an Arduino Leonardo processor, you can build yourself a fun and responsive little bot that will chase objects or follow you around like a playful pet!

The Pixy Pet Robot is simple to build with no soldering required. With just a few common tools, you can complete the assembly in under an hour!

Before embarking on this project, please follow the Pixy and Zumo tutorials, getting those working with the Arduino seperately and then you can combine them!



Materials:

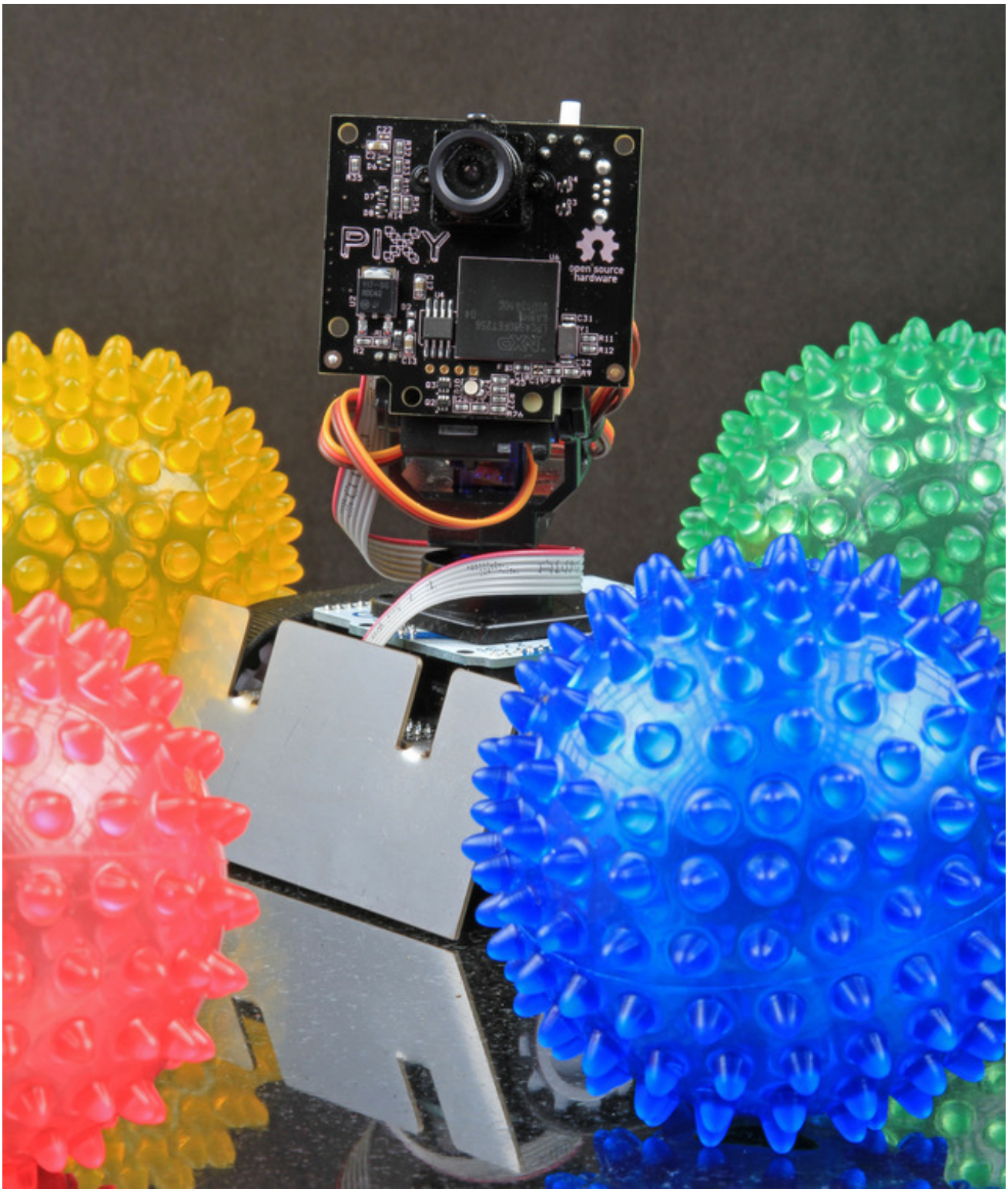
- Pixy CMUcam-5 (http://adafru.it/dSP)
- Mini Pan/Tilt Kit - Assembled with Micro Servos (http://adafru.it/1967)*
- Zumo Robot (http://adafru.it/dSQ)
- Arduino Leonardo (http://adafru.it/849)
- Double-sided foam tape
- Cable Ties
- 4x AA batteries

* If you have some micro-servos already, we also have an unassembled pan/tilt kit (http://adafru.it/1968) in the store. Some modifications may be required to fit your servo horns to the pan/tilt kit.

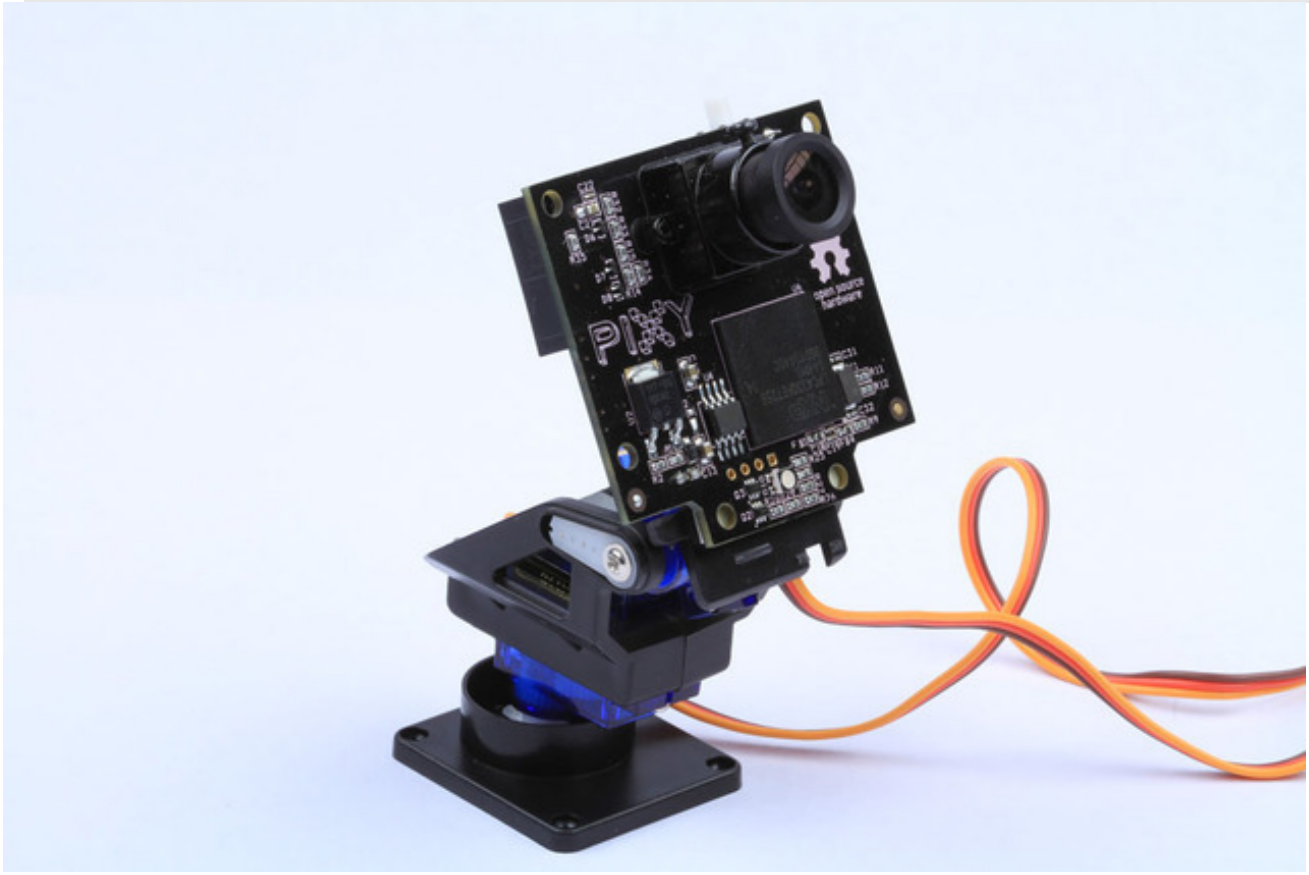
Tools:

- Wire Cutters
- Scissors
- USB A to Mini-B cable (for teaching the camera)
- USB A to Micro-B cable (for uploading to the Leonardo)

Note: Due to the pin assignments of the Zumo robot shield, this project will not work with an Uno or other Atmega 328-based processor.



Assemble the Camera

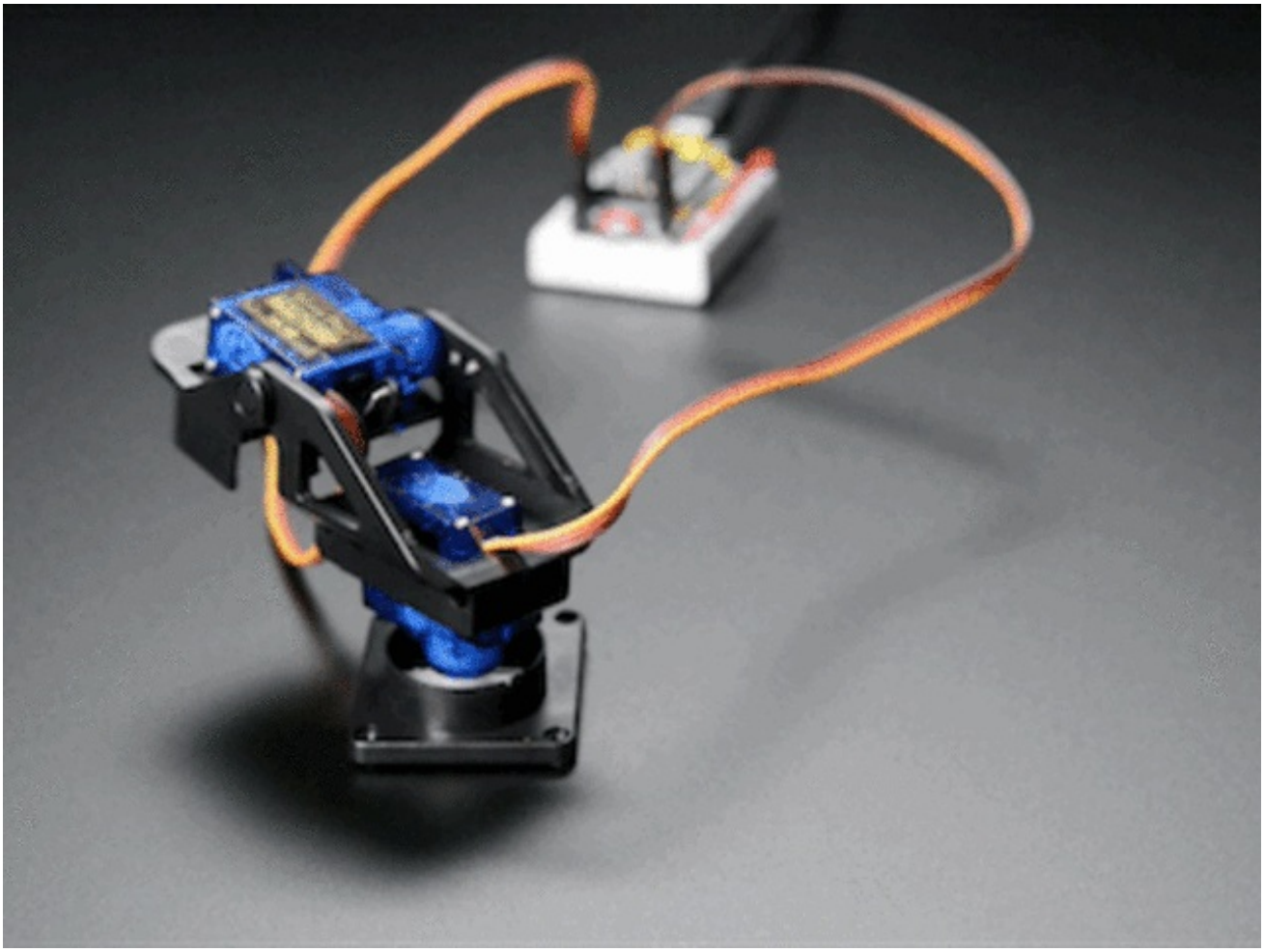


The Pixy Camera itself is fully assembled. We just need to attach it to the pan/tilt base and connect the servos.

The pan/tilt kit is available with and without sevos. The mechanism is sized to fit standard micro sevo cases. However, servo horns are not standardized. If using other servos, you may need to trim or re-shape the horns to fit.

Preparing the Pan/Tlt Base

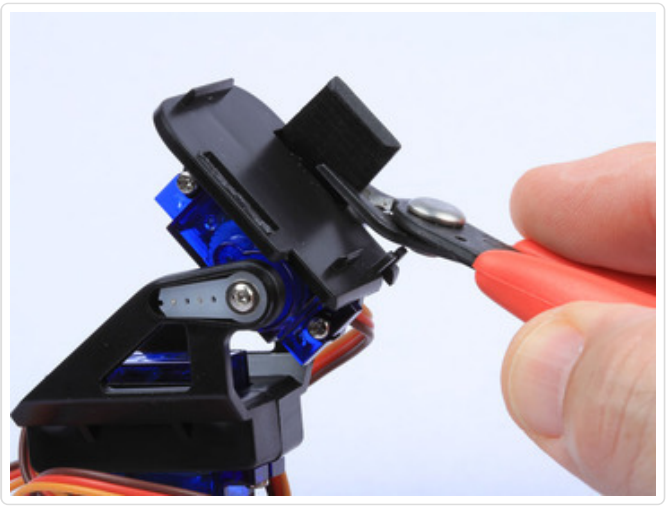
The pan/tilt base has mounting tabs for a different style of camera module. We'll need to remove these before attaching to the Pixy CMU-Cam.



Remove the Side Tabs

Cut the side tabs so that they are flush with the face of the camera mounting bracket.

The bracket is made of a fairly soft nylon, so these are easly removed with a pair of wire cutters.

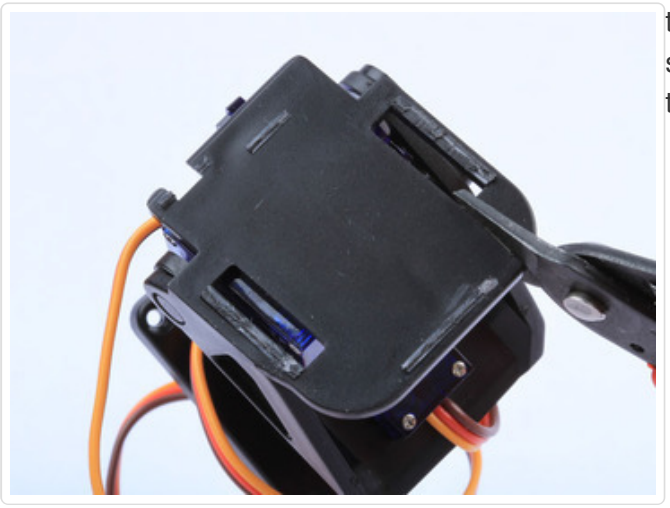


Trim the remaining tabs flush

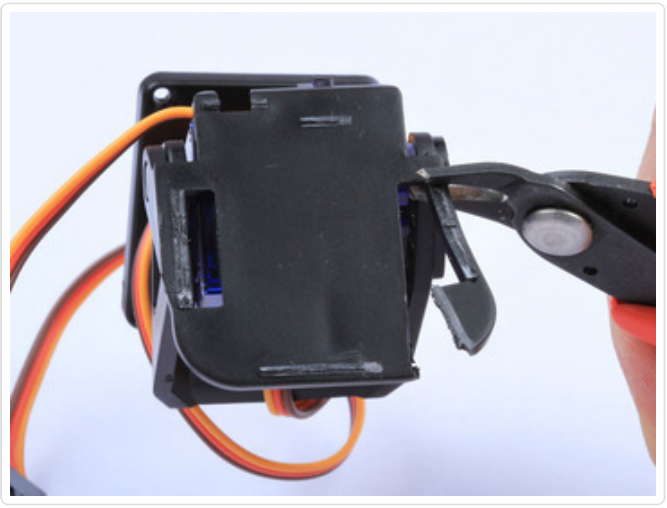
There are two smaller alignment tabs and a cable guide that must be trimmed flush also.

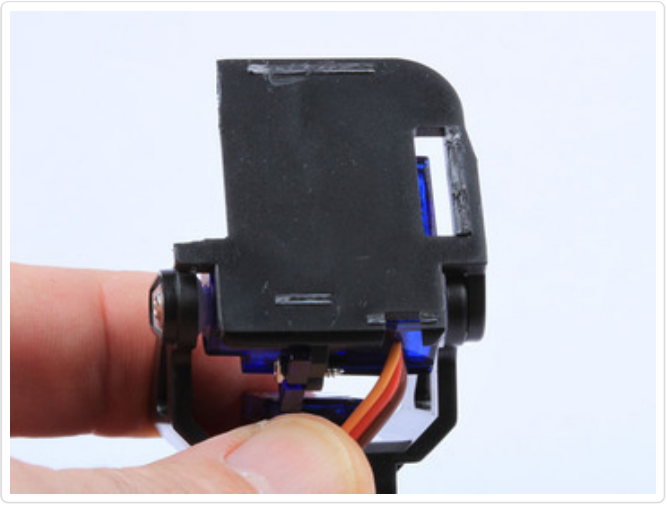
Cut a notch for the cable connectors

We need to make room for the cable connectors on



the back of the camera module. Two cuts, as shown in the photos will remove the top left side of the bracket.



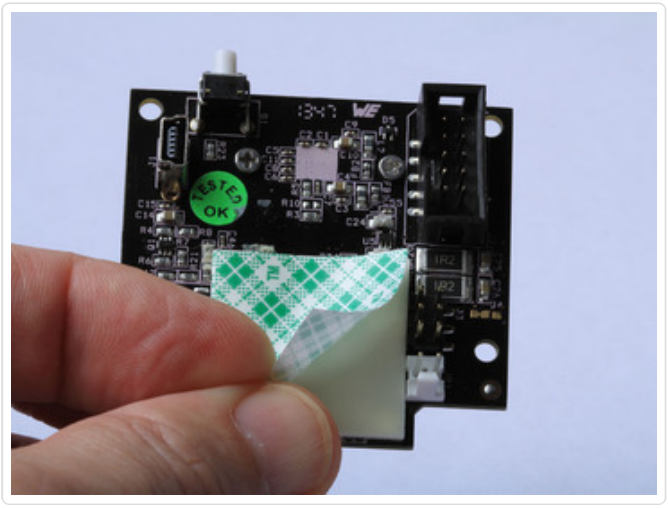


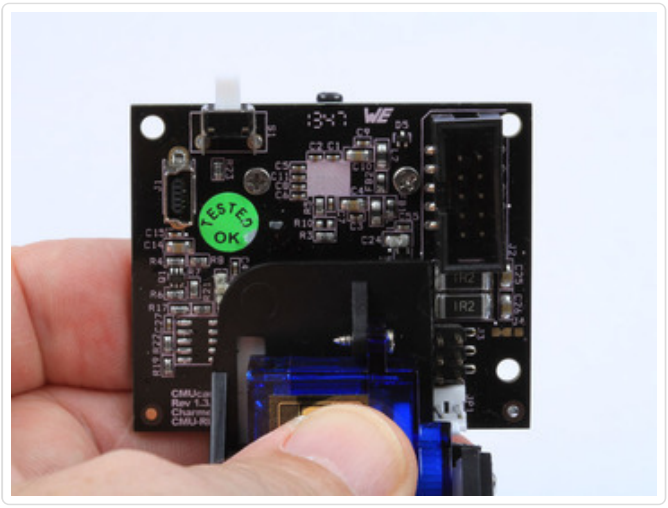
Attach the Camera

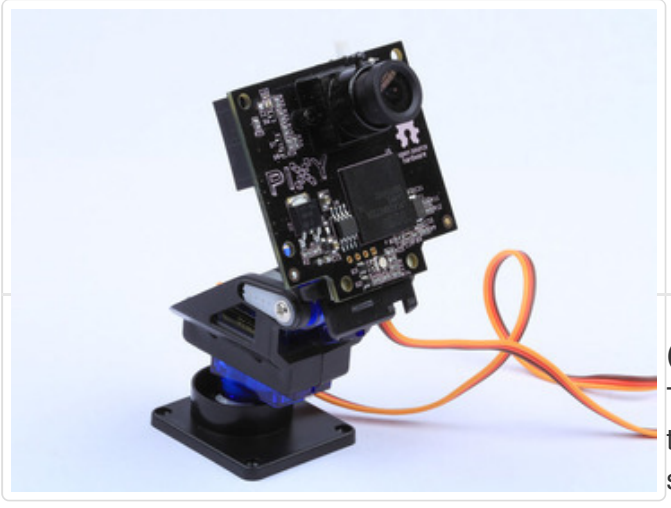
Cut a 1"x1" piece of double-sided foam tape and position on the back of the camera module as shown.

Remove the backing paper and align the camera bracket as shown. The connector headers on the back of the camera module will fit into the notch we cut in the previous step.

Press down firmly to adhere the camera to the mounting bracket. Your final assembly should look like the last photo to the left..



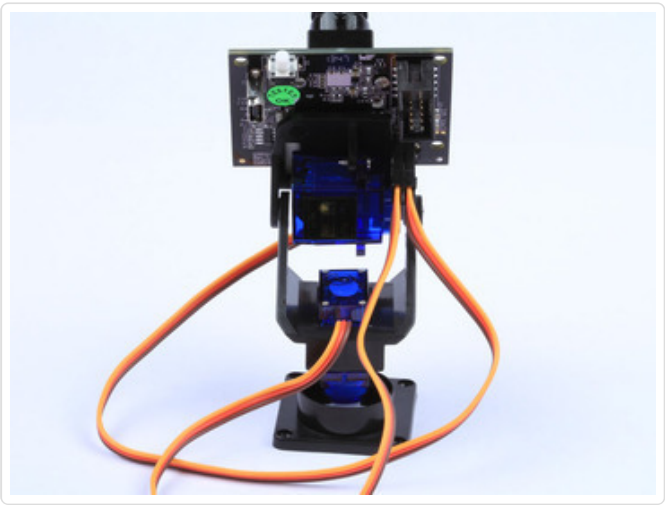




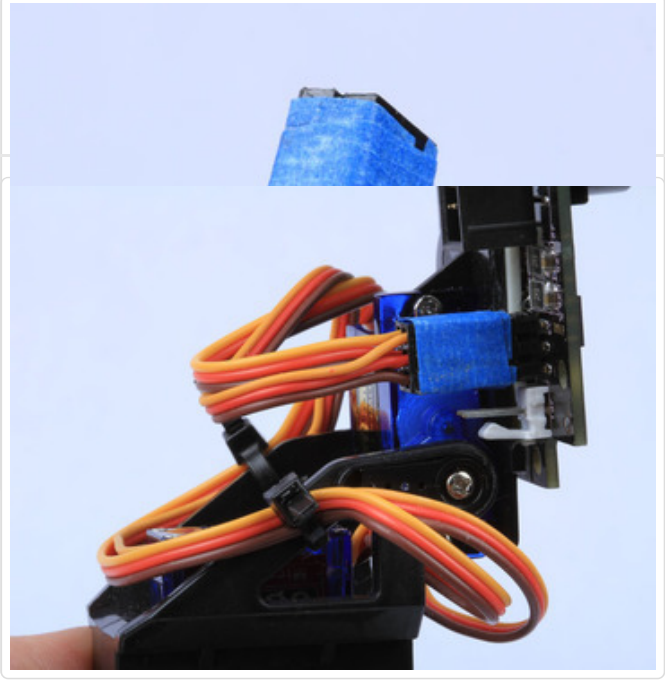
Connect the Servo Cables

The servo cables attach to the 2x6 pin header on the back of the camera. The cable for the pan servo (the bottom one) should be on the left. The cable for the tilt servo should be on the right. Make sure that the brown wire is on the bottom and the yellow wire is on top.

Tape the two connectors together. This will make it

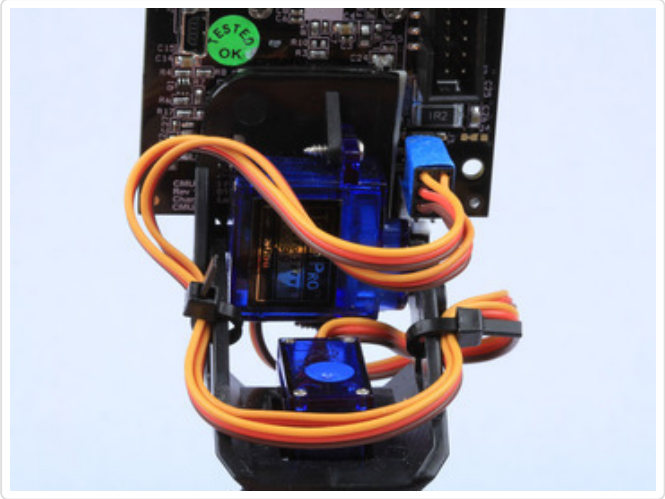


easier to keep them from getting mixed up if you have to disconnect them later.



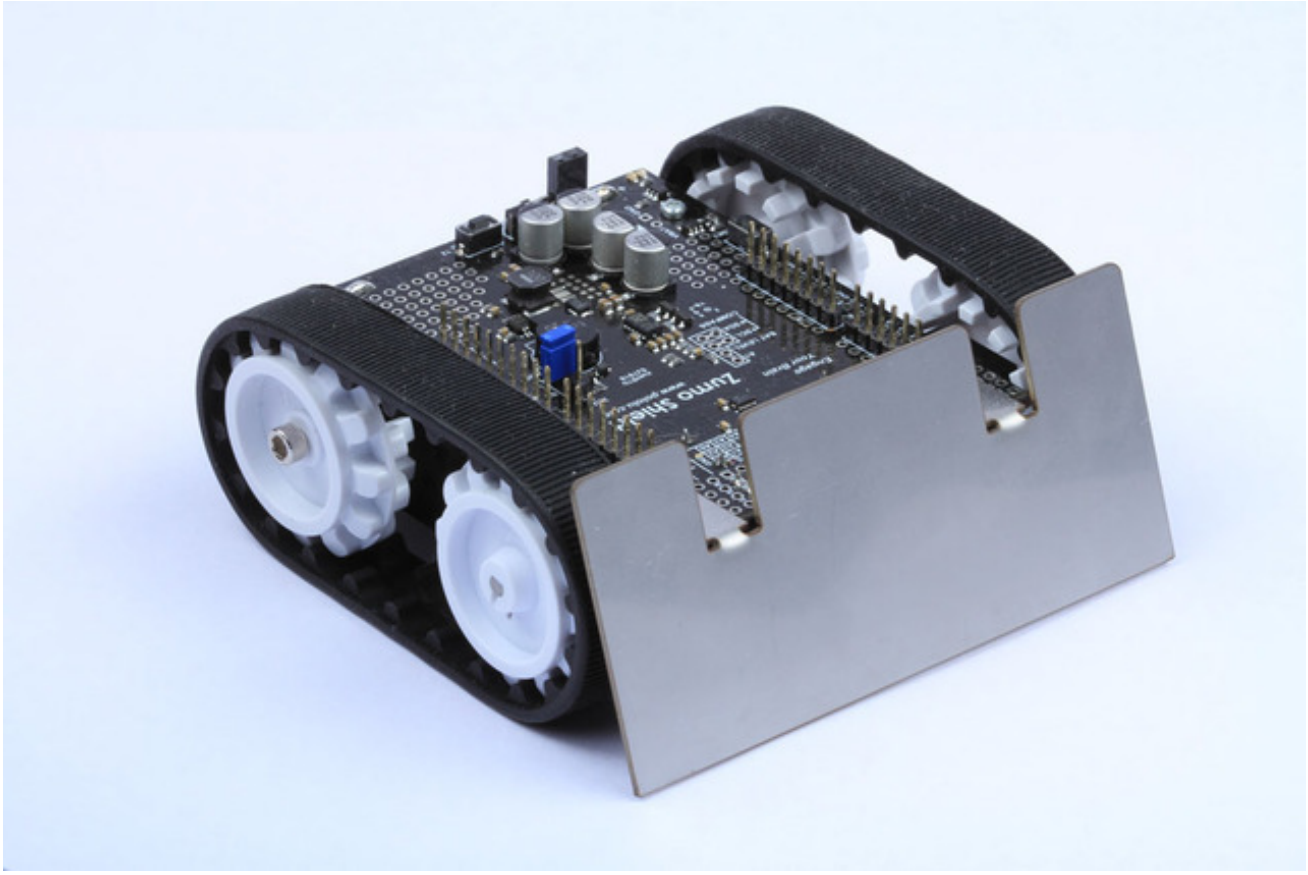
Secure the Cables

Route the servo cables as shown and anchor to the pan/tilt base with cable ties. Be sure to leave enough slack so that the cables will not interfere with the pan/tlt motion.



Final Assembly

The Zumo itself comes pre-assembled, minus the Leonardo processor. We just need to attach the processor and camera assembly and connect the cables.

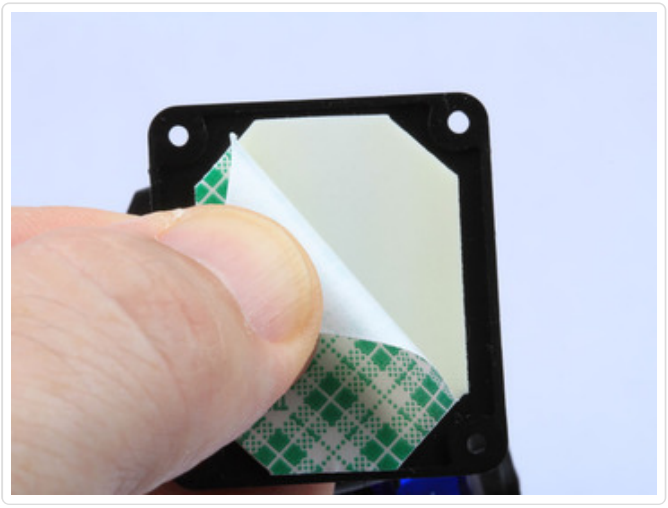


Attach the Camera to the Leonardo

Cut a piece of foam tape to fit in the recess in the bottom of the pan/tilt base. (If you are using narrower tape, you can use multiple pieces.)

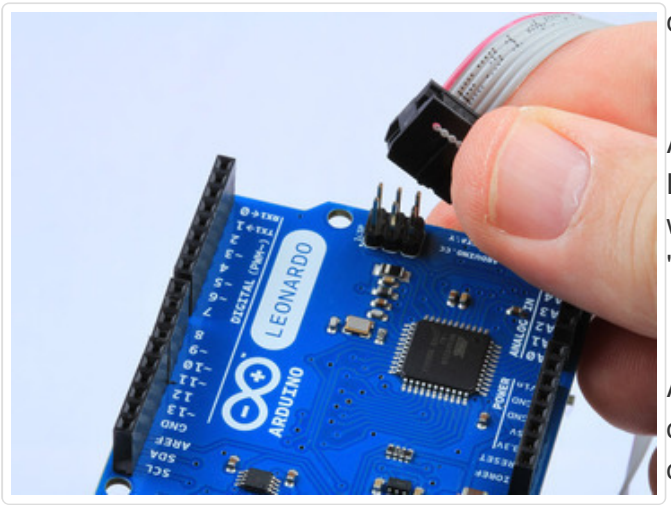
Position the camera as shown on the bottom of the Leonardo and press firmly to attach.

You can use the dotted line above the FCC/CE logos for alignment!



Connect the ribbon cable

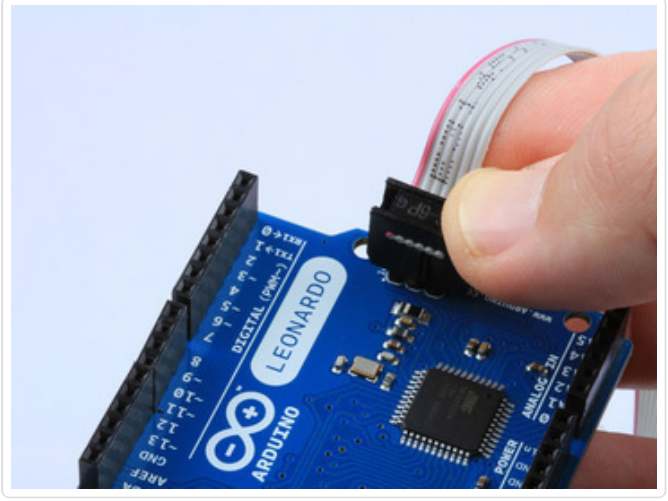
The gray ribbon cable that came with your Pixy has

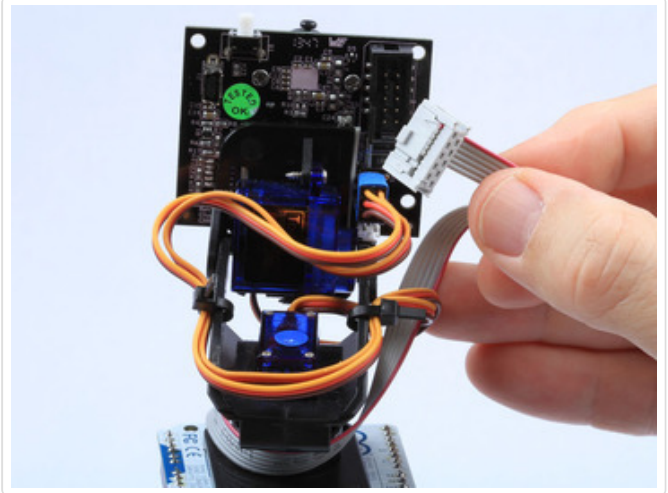


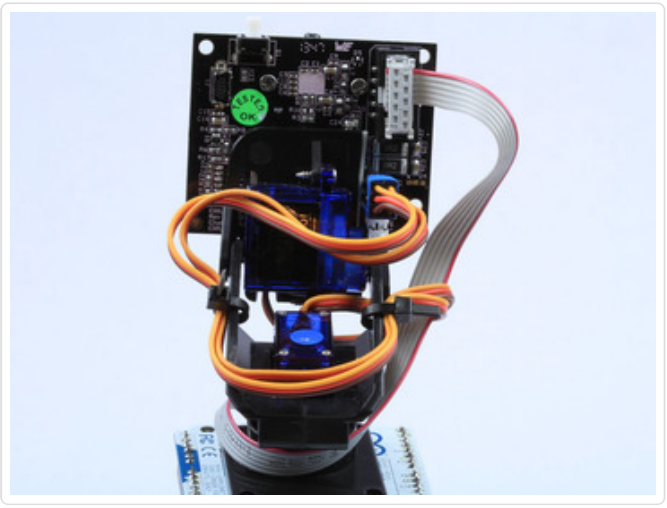
one 6-pin connector and one 8-pin connector.

Attach the 6-pin end to the ICSP header on the Leonardo as shown. Make sure to align the edge with the red-stripe so that it is closest to the "LEONARDO" logo on the board.

Attach the 8-pin end to the back of the Pixy. This connection is keyed, so there is only one way you can plug it in.



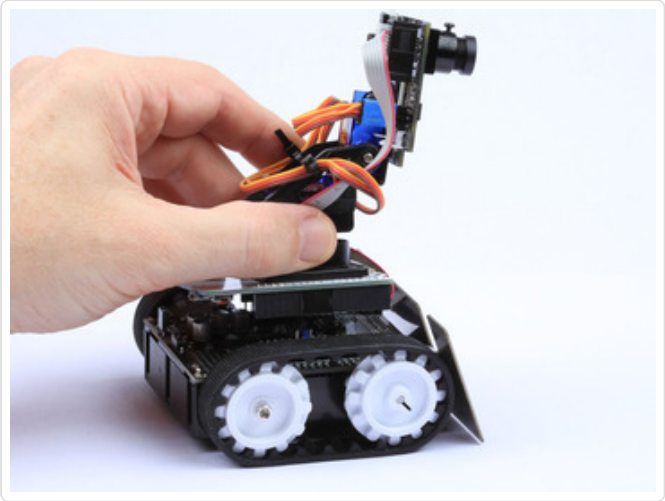


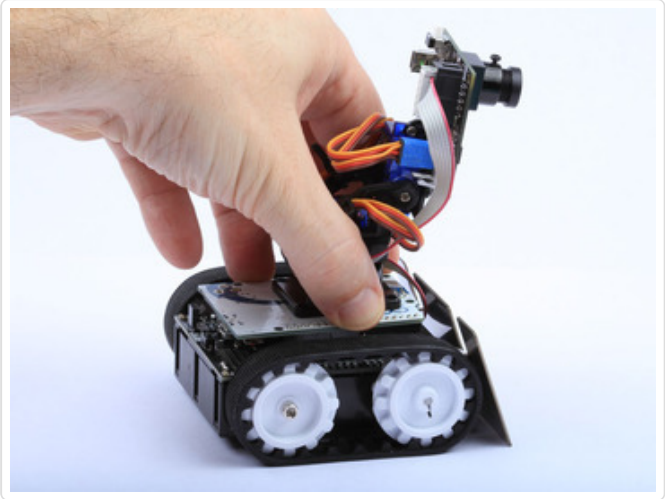


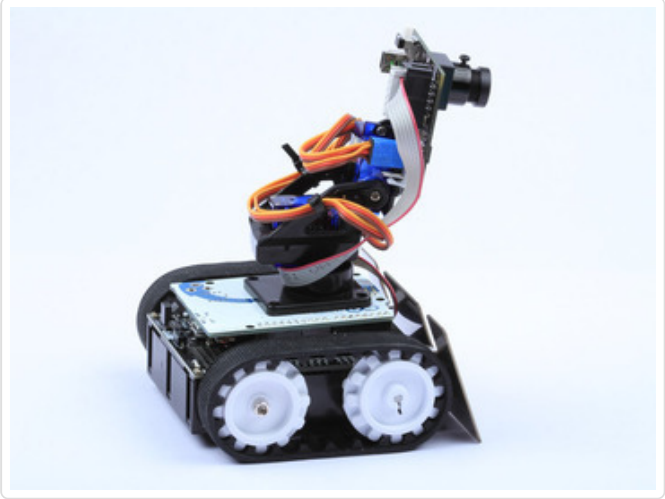
Attach the Camera and Processor to the Zumo

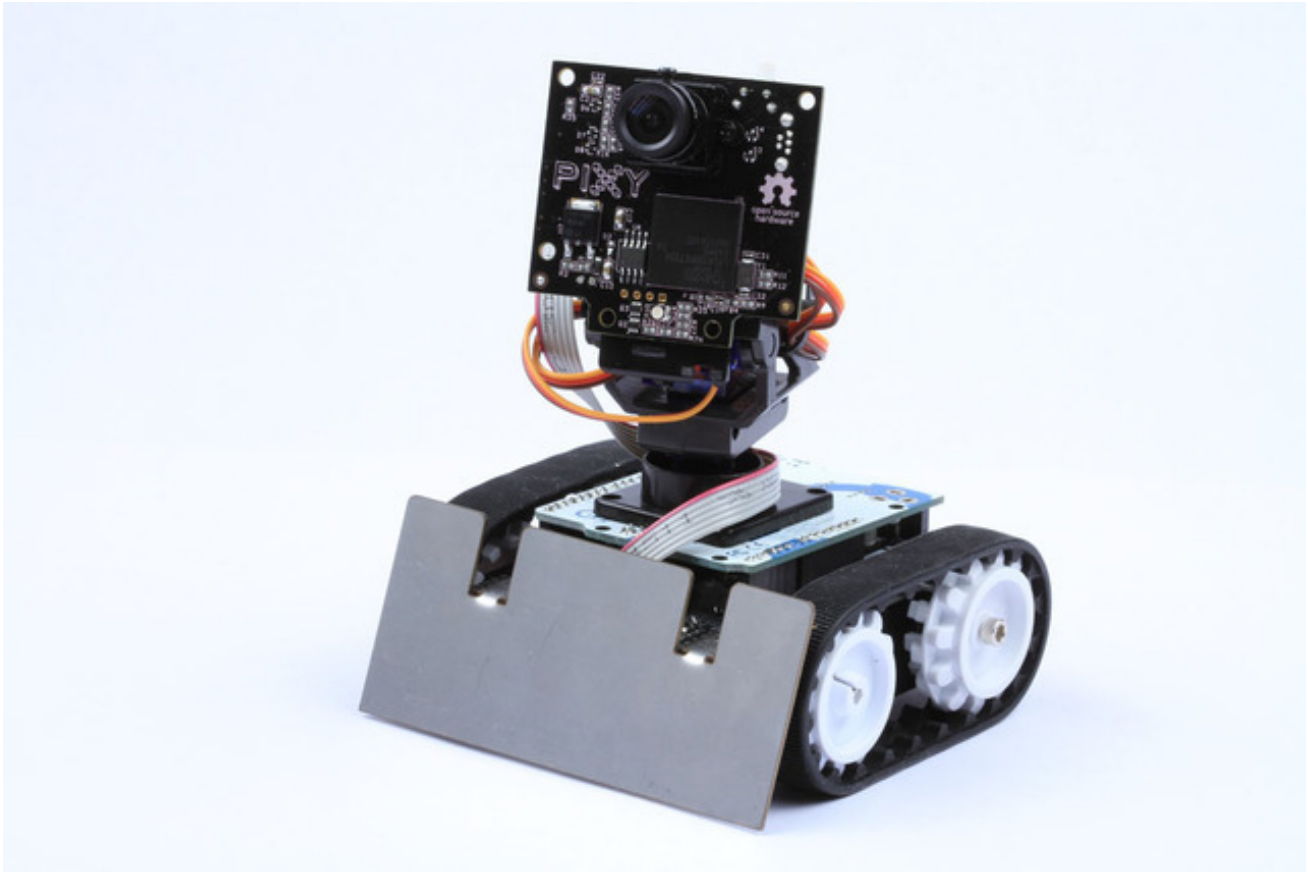
Align the Leonardo with the header pins on the Zumo. The camera should be facing the front.

Press firmly to seat the board on the headers. And you are done!





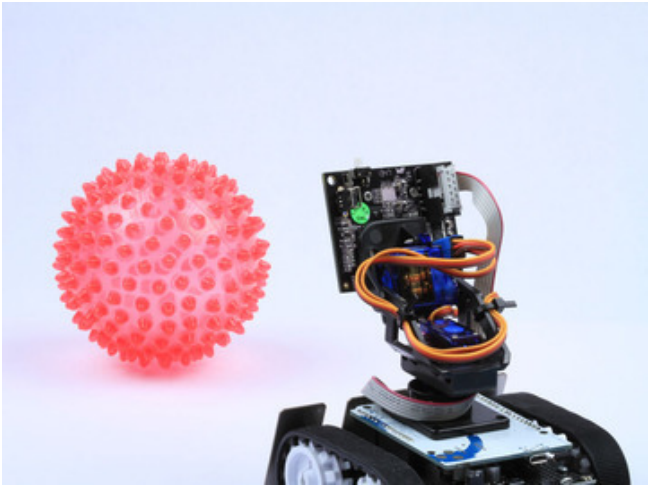



PIXY

Playing with your Pixy Pet!

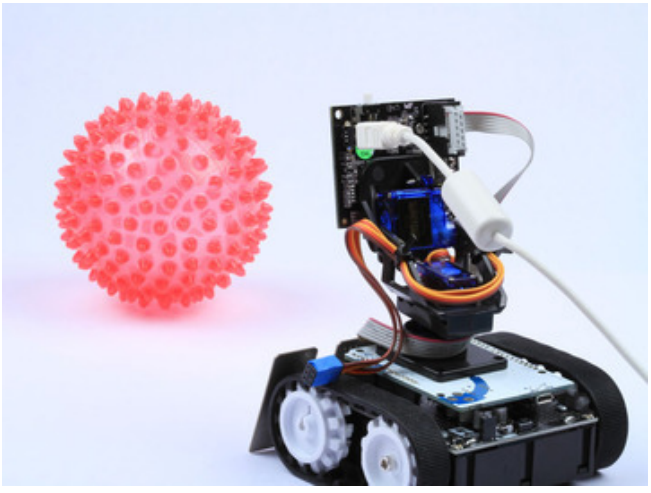
Teach the Camera

The first thing you need to do is teach Pixy the objects you want it to track. The best way to do this is using the PixyMon software. With PixyMon, you can see exactly what Pixy sees and how well it has learned.



Find a toy!

Brightly colored balls are good. Place it in view of the Pixy camera.



Connect the Camera

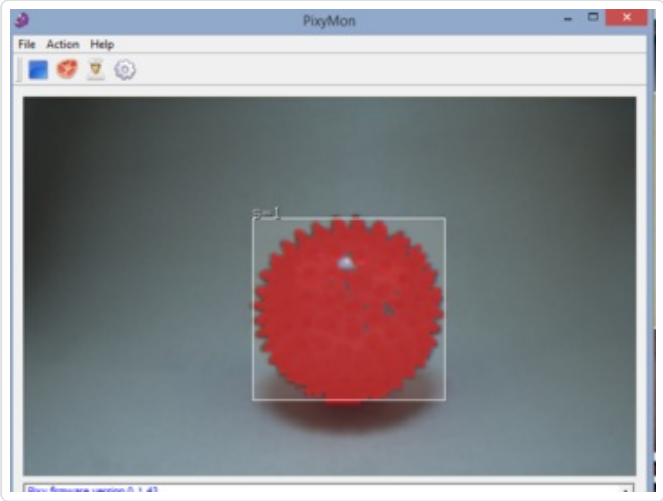
Connect the camera to your computer using a mini-B USB cable.

Hint: *If the software has already been loaded, it helps to disconnect the servos during the teaching process.*

Download PixyMon

http://adafru.it/dSR

Run PixyMon



- Download PixyMon from the link above. (*There are several versions, be sure to pick the right one for your operating system.)*
- Launch the PixyMon application.
- Select the "Cooked" view (*click on the icon with the chef's hat!*) This view will show you exactly what the Pixy camera sees in real-time.
- Click "**Action->Set Signature1...**"
- Select an area on the ball to teach a color to the camera.

Once learned, the camera will indicate recognized objects with a rectangle and signature number.

Upload the Code

Open the Arduino IDE and load the Pixy Code from the following page. Connect a USB cable to the Leonardo on the Pixy Pet and upload the code.

Note: If the upload fails, try pressing the reseet button on the left side of the Zumo board, shortly before the compile completes.

Play Ball!

- Disconnect all the USB cables and make sure that the Servo cables are plugged into the camera.
- Make sure that the batteries are installed in the Zumo robot base.
- Turn on the Zumo using the on/off switch located at the rear of the Zumo.

Once the bootloader has finished (the yellow led will stop flashing), Pixy will start looking for the ball. Once it sees the ball it will move toward it and start to follow it around.

The Code

Copy the code below into the Arduino IDE and upload. Make sure that you have selected "Arduino Leonardo" in Tools->Board:

```
//============================================================================
//
//  Pixy Pet Robot
//
//   Adafruit invests time and resources providing this open source code,
//  please support Adafruit and open-source hardware by purchasing
//  products from Adafruit!
//
// Written by: Bill Earl for Adafruit Industries
//
//============================================================================
// begin license header
//
// All Pixy Pet source code is provided under the terms of the
// GNU General Public License v2 (http://www.gnu.org/licenses/gpl-2.0.html).
//
// end license header
//
//============================================================================
//
// Portions of this code are derived from the Pixy CMUcam5 pantilt example code.
//
//============================================================================
#include <SPI.h>
#include <Pixy.h>

#include <ZumoMotors.h>

#define X_CENTER    160L
#define Y_CENTER    100L
#define RCS_MIN_POS     0L
#define RCS_MAX_POS     1000L
#define RCS_CENTER_POS ((RCS_MAX_POS-RCS_MIN_POS)/2)

//---------------------------------------
// Servo Loop Class
// A Proportional/Derivative feedback
// loop for pan/tilt servo tracking of
// blocks.
// (Based on Pixy CMUcam5 example code)
//---------------------------------------
```

```
class ServoLoop
{
public:
 ServoLoop(int32_t proportionalGain, int32_t derivativeGain);

 void update(int32_t error);

 int32_t m_pos;
 int32_t m_prevError;
 int32_t m_proportionalGain;
 int32_t m_derivativeGain;
};

// ServoLoop Constructor
ServoLoop::ServoLoop(int32_t proportionalGain, int32_t derivativeGain)
{
 m_pos = RCS_CENTER_POS;
 m_proportionalGain = proportionalGain;
 m_derivativeGain = derivativeGain;
 m_prevError = 0x80000000L;
}

// ServoLoop Update
// Calculates new output based on the measured
// error and the current state.
void ServoLoop::update(int32_t error)
{
 long int velocity;
 char buf[32];
 if (m_prevError!=0x80000000)
 {
  velocity = (error*m_proportionalGain + (error - m_prevError)*m_derivativeGain)>>10;

  m_pos += velocity;
  if (m_pos>RCS_MAX_POS)
  {
   m_pos = RCS_MAX_POS;
  }
  else if (m_pos<RCS_MIN_POS)
  {
   m_pos = RCS_MIN_POS;
  }
 }
 m_prevError = error;
}
// End Servo Loop Class
//-------------------------------------
```

```
//

Pixy pixy;  // Declare the camera object

ServoLoop panLoop(200, 200);  // Servo loop for pan
ServoLoop tiltLoop(150, 200); // Servo loop for tilt

ZumoMotors motors;  // declare the motors on the zumo

//---------------------------------------
// Setup - runs once at startup
//---------------------------------------
void setup()
{
 Serial.begin(9600);
 Serial.print("Starting...\n");

 pixy.init();
}

uint32_t lastBlockTime = 0;

//---------------------------------------
// Main loop - runs continuously after setup
//---------------------------------------
void loop()
{
 uint16_t blocks;
 blocks = pixy.getBlocks();

 // If we have blocks in sight, track and follow them
 if (blocks)
 {
  int trackedBlock = TrackBlock(blocks);
  FollowBlock(trackedBlock);
  lastBlockTime = millis();
 }
 else if (millis() - lastBlockTime > 100)
 {
  motors.setLeftSpeed(0);
  motors.setRightSpeed(0);
  ScanForBlocks();
 }
}

int oldX, oldY, oldSignature;

//
```

```
//--------------------------------------
// Track blocks via the Pixy pan/tilt mech
// (based in part on Pixy CMUcam5 pantilt example)
//--------------------------------------
int TrackBlock(int blockCount)
{
 int trackedBlock = 0;
 long maxSize = 0;

 Serial.print("blocks =");
 Serial.println(blockCount);

 for (int i = 0; i < blockCount; i++)
 {
  if ((oldSignature == 0) || (pixy.blocks[i].signature == oldSignature))
  {
   long newSize = pixy.blocks[i].height * pixy.blocks[i].width;
   if (newSize > maxSize)
   {
    trackedBlock = i;
    maxSize = newSize;
   }
  }
 }

 int32_t panError = X_CENTER - pixy.blocks[trackedBlock].x;
 int32_t tiltError = pixy.blocks[trackedBlock].y - Y_CENTER;

 panLoop.update(panError);
 tiltLoop.update(tiltError);

 pixy.setServos(panLoop.m_pos, tiltLoop.m_pos);

 oldX = pixy.blocks[trackedBlock].x;
 oldY = pixy.blocks[trackedBlock].y;
 oldSignature = pixy.blocks[trackedBlock].signature;
 return trackedBlock;
}

//--------------------------------------
// Follow blocks via the Zumo robot drive
//
// This code makes the robot base turn
// and move to follow the pan/tilt tracking
// of the head.
//--------------------------------------
int32_t size = 400;
```

```
void FollowBlock(int trackedBlock)
{
 int32_t followError = RCS_CENTER_POS - panLoop.m_pos;  // How far off-center are we looking now?

 // Size is the area of the object.
 // We keep a running average of the last 8.
 size += pixy.blocks[trackedBlock].width * pixy.blocks[trackedBlock].height;
 size -= size >> 3;

 // Forward speed decreases as we approach the object (size is larger)
 int forwardSpeed = constrain(400 - (size/256), -100, 400);

 // Steering differential is proportional to the error times the forward speed
 int32_t differential = (followError + (followError * forwardSpeed))>>8;

 // Adjust the left and right speeds by the steering differential.
 int leftSpeed = constrain(forwardSpeed + differential, -400, 400);
 int rightSpeed = constrain(forwardSpeed - differential, -400, 400);

 // And set the motor speeds
 motors.setLeftSpeed(leftSpeed);
 motors.setRightSpeed(rightSpeed);
}

//---------------------------------------
// Random search for blocks
//
// This code pans back and forth at random
// until a block is detected
//---------------------------------------
int scanIncrement = (RCS_MAX_POS - RCS_MIN_POS) / 150;
uint32_t lastMove = 0;

void ScanForBlocks()
{
 if (millis() - lastMove > 20)
 {
  lastMove = millis();
  panLoop.m_pos += scanIncrement;
  if ((panLoop.m_pos >= RCS_MAX_POS)||(panLoop.m_pos <= RCS_MIN_POS))
  {
   tiltLoop.m_pos = random(RCS_MAX_POS * 0.6, RCS_MAX_POS);
   scanIncrement = -scanIncrement;
   if (scanIncrement < 0)
   {
    motors.setLeftSpeed(-250);
    motors.setRightSpeed(250);
```

```
    }
    else
    {
     motors.setLeftSpeed(+180);
     motors.setRightSpeed(-180);
    }
    delay(random(250, 500));
   }

   pixy.setServos(panLoop.m_pos, tiltLoop.m_pos);
  }
}
```

Pixy Pet Code Design

OK. That was fun, but how does it work?

The Pixy Robot code consists of two main control systems: **Object Tracking** with the Pixy Camera and the pan/tilt mechanism and **Object Following** with the Zumo robot base.

Together these two systems produce a very natural looking response where the 'head' turns in response to motion and the 'body' follows.

Both control systems are based on **Feedback Control Loops**. For a detailed explanation of how Feedback Control works, see the **Feedback Control Basics** page in this guide.

Tracking Objects

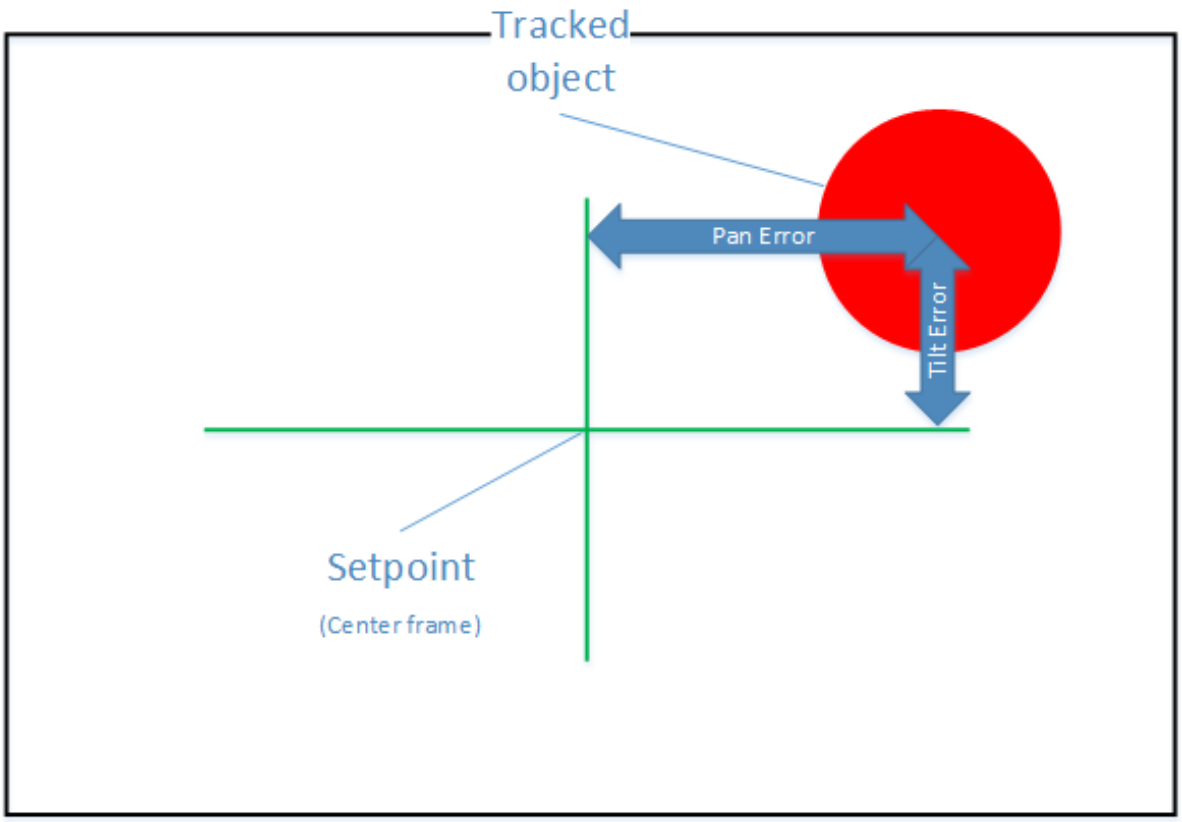
Object tracking is implemented in the TrackBlock function. The hard work of object detection and location is handled by the image processing system inside the Pixy camera. It analyzes the image and identifies objects matching the color characteristics of the object being tracked. It then reports the position size and colors of all the detected objects back to the Arduino.

In the Arduino, we use this information to adjust the pan and tilt servos to try to keep the tracked object in the center of the field of view.

```
//--------------------------------------
// Track blocks via the Pixy pan/tilt mech
// (based in part on Pixy CMUcam5 pantilt example)
//--------------------------------------
int TrackBlock(int blockCount)
{
 int trackedBlock = 0;
 long maxSize = 0;

 Serial.print("blocks =");
 Serial.println(blockCount);

 for (int i = 0; i < blockCount; i++)
 {
  if ((oldSignature == 0) || (pixy.blocks[i].signature == oldSignature))
  {
   long newSize = pixy.blocks[i].height * pixy.blocks[i].width;
   if (newSize > maxSize)
   {
    trackedBlock = i;
    maxSize = newSize;
   }
  }
 }

 int32_t panError = X_CENTER - pixy.blocks[trackedBlock].x;
 int32_t tiltError = pixy.blocks[trackedBlock].y - Y_CENTER;

 panLoop.update(panError);
 tiltLoop.update(tiltError);

 pixy.setServos(panLoop.m_pos, tiltLoop.m_pos);

 oldX = pixy.blocks[trackedBlock].x;
 oldY = pixy.blocks[trackedBlock].y;
 oldSignature = pixy.blocks[trackedBlock].signature;
 return trackedBlock;
}
```

The Pan/Tilt control is implemented using 2 instances of the ServoLoop class - one for the pan and one for the tilt. ServoLoop is a feedback control loop using both Proportional + Derivative (PD) control. The **measurements** are the x (for pan) and y (for tilt) positions of the blocks reported by the Pixy Camera. The **setpoints** are the x, y position of the center of the camera's view. And the **outputs** are the servo positions.

On each pass through the main loop, we calculate the errors for the pan and tilt controls as the difference between the measurements and the setpoints. Then we invoke the ServoLoop control algorithms to calculate the outputs.




```
//--------------------------------------
// Servo Loop Class
// A Proportional/Derivative feedback
// loop for pan/tilt servo tracking of
// blocks.
// (Based on Pixy CMUcam5 example code)
//--------------------------------------
class ServoLoop
{
public:
 ServoLoop(int32_t proportionalGain, int32_t derivativeGain);

 void update(int32_t error);

 int32_t m_pos;
 int32_t m_prevError;
 int32_t m_proportionalGain;
 int32_t m_derivativeGain;
```

```
};

// ServoLoop Constructor
ServoLoop::ServoLoop(int32_t proportionalGain, int32_t derivativeGain)
{
 m_pos = RCS_CENTER_POS;
 m_proportionalGain = proportionalGain;
 m_derivativeGain = derivativeGain;
 m_prevError = 0x80000000L;
}

// ServoLoop Update
// Calculates new output based on the measured
// error and the current state.
void ServoLoop::update(int32_t error)
{
 long int velocity;
 char buf[32];
 if (m_prevError!=0x80000000)
 {
  velocity = (error*m_proportionalGain + (error - m_prevError)*m_derivativeGain)>>10;

  m_pos += velocity;
  if (m_pos>RCS_MAX_POS)
  {
   m_pos = RCS_MAX_POS;
  }
  else if (m_pos<RCS_MIN_POS)
  {
   m_pos = RCS_MIN_POS;
  }
 }
 m_prevError = error;
}
// End Servo Loop Class
//---------------------------------------
```

Following Objects

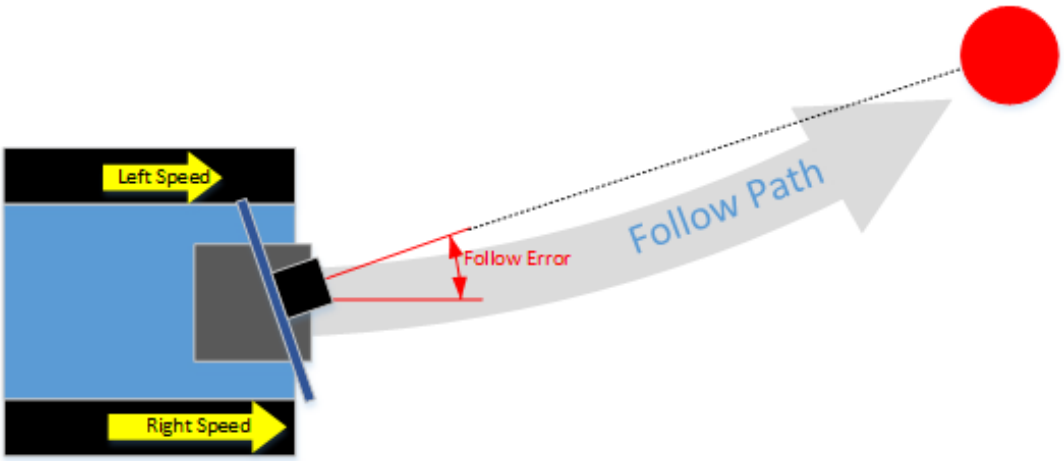
The object following behavior is implemented in the FollowBlock function. FollowBlock uses just proportional control. But we have two measurements (size and pan position) and two outputs (left and right drive motors).

The size (block height times width) gives us a rough idea of how far away the object is and we use that to calculate the 'forwardSpeed'. This makes the robot slow down as it approaches the object. If the object appears larger than the setpoint value, forwardSpeed will become negative and the robot

will back up.

```
//--------------------------------------
// Follow blocks via the Zumo robot drive
//
// This code makes the robot base turn
// and move to follow the pan/tilt tracking
// of the head.
//--------------------------------------
int32_t size = 400;
void FollowBlock(int trackedBlock)
{
 int32_t followError = RCS_CENTER_POS - panLoop.m_pos;  // How far off-center are we looking now?

 // Size is the area of the object.
 // We keep a running average of the last 8.
 size += pixy.blocks[trackedBlock].width * pixy.blocks[trackedBlock].height;
 size -= size >> 3;

 // Forward speed decreases as we approach the object (size is larger)
 int forwardSpeed = constrain(400 - (size/256), -100, 400);

 // Steering differential is proportional to the error times the forward speed
 int32_t differential = (followError + (followError * forwardSpeed))>>8;

 // Adjust the left and right speeds by the steering differential.
 int leftSpeed = constrain(forwardSpeed + differential, -400, 400);
 int rightSpeed = constrain(forwardSpeed - differential, -400, 400);

 // And set the motor speeds
 motors.setLeftSpeed(leftSpeed);
 motors.setRightSpeed(rightSpeed);
}
```

The pan position (one of the outputs of the tracking control) tells us how far the head is turned away from the setpoint (straight-ahead). This value is used to control the speed differential between the left and right motors - causing the robot to turn toward the object it is following.


Left Speed
Right Speed
Follow Error
Follow Path

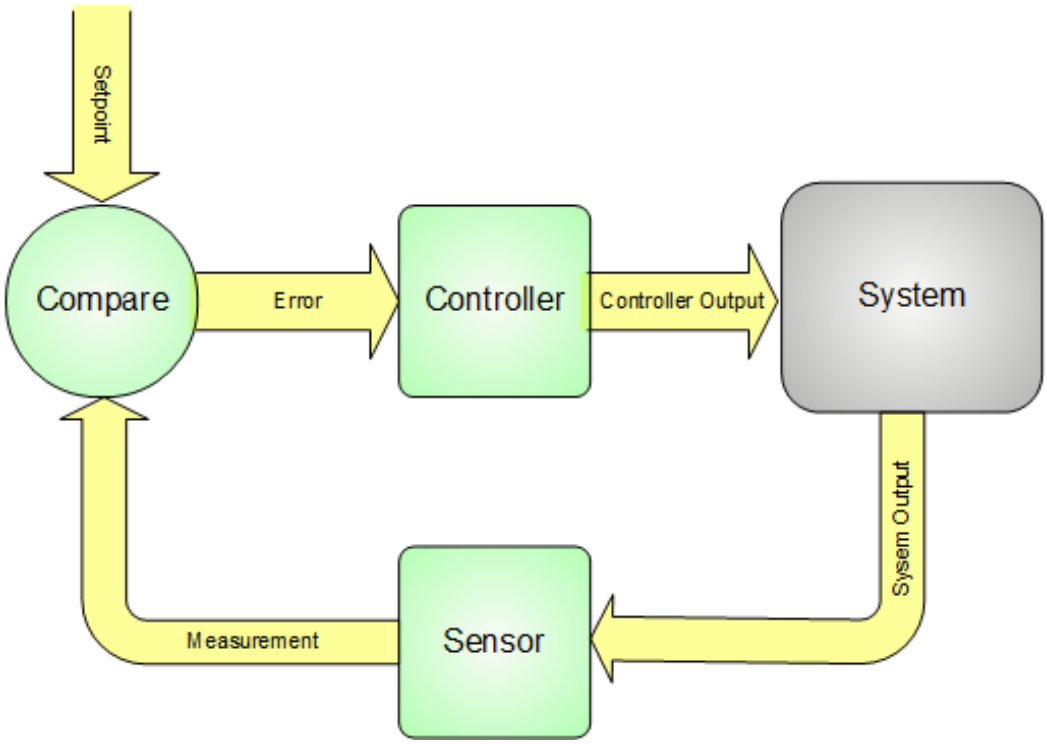
Feedback Control Basics

Measurements, Setpoints, Errors and Ouputs

To start with, let's define some terms commonly used to describe control systems:

- **Measurement** - This is typically the value of the parameter you are trying to control. It could be temperature, pressure, speed, position or any other parameter. Before you can control anything, you have to be able to measure it.
- **Setpoint** - This is the desired value for the parameter you are trying to control.
- **Error** - This is the difference between the desired value and the measured value.
- **Output** - This is a value calculated based on the error. It is fed back into the system to 'correct' the error and bring the measurement closer to the setpoint.

There are many ways in which the output value can be calculated. We'll discuss a few of the more common ones.



Types of Control

There are many ways in which the output value can be calculated. We'll discuss a few of the more common ones.

On/Off Control

In this type of control, the only values for the **output** are ON or OFF. This is how the thermostat in your house works. If the **measured** temperature is below the **setpoint** temperature, it turns on the

heat. If the **measured** temperature is above the **setpoint**, it turns it off. To prevent rapid cycling which could damage the system, there is typically some gap between the 'on' threshold and the 'off' threshold. This is called 'hysteresis'.

An On/Off controller with hysteresis s sometimes called a "Differential Gap Controller". That sounds pretty sophisticated, but it is still a very primitive type of controller.

On/Off control works well for controlling the temperature of your house, but it is not very good for applications like robot motion control.

PID Control

You have probably heard of **PID** controllers. **PID** stands for **P**roportional, **I**ntegral and **D**erivative control. So a PID controller is actually 3 types of controller in one. Because of this, PID control is fairly versatile. But not all applications require all three forms of control.

Many so-called PID controllers are actually just operated as **PI**, **PD** or even just **P** type controllers. Motion Control applications like the Pixy Pet generally use mostly **P** or **PD** control.

Proportional Control

Proportional control allows for a much smoother response than simple on/off control. Proportional control calculates an **output** value that is proportional to the magnitude of the **error**. Small errors yeild a small response. Larger errors result in a more aggressive response.

Proportional control can be used alone, or augmented with Integral or Derivative control as needed. The Pixy object following code uses only proportional control. The object tracking code uses both proportional and derivative control.

Integral Control

Integral control integrates the error over time. If the measurement is not converging on the setpoint, the integral output keeps increasing to drive the system toward the setpoint.

Integral control is good for nudging steady, predictable processes closer to perfection. Since Pixy Pet needs to always respond quickly to random unpredictable movements, integral control is not appropriate.

Derivative Control

Derivative control looks at the rate of change in the error. If the error is rapidly approaching zero, the output of the derivative calculation attempts to slow things down to avoid overshooting the

setpoint.

The Pixy object tracking algorithm uses derivative control in conjunction with the proportional control to help prevent over-correction when tracking objects.

Troubleshooting

Pixy Pet wont track an object

Pixy Pet tracks best if the object is a bright saturated color. It also helps if there are not a lot of similarly colored things in the environment to distract your Pixy.

Pixy Pet loses the tracked object - even when it is right in front of it.

Pixy Pet performs best in a brightly lit area. Check with PixyMon to make sure that Pixy recognizes the object well and re-teach that color signature if necessary.

Sometimes, moving to an area with different lighting (e.g. daylight vs. flourescent) can change the color appearance and confuse Pixy.

Pixy Pet is easily distracted by other objects

Other objects of the same color can distract Pixy Pet if they are in view. Pixy Pet will tend to favor the largest recognized object. Teaching Pixy Pet too many different color signatures increases the chances for confusion. It is better to stick to one color at a time.

Pixy Pet moves erratically when the object is in view, but doesn't track it

Make sure your batteries are fresh. Make sure you don't have the pan and tilt servo plugs reversed.

Pixy Pet's pan/tilt head oscillates - even when the object is still

Reduce the proportional gain in the ServoLoops.

Pixy Pet seems sluggish and the pan/tilt tracking keeps glitching.

Your Pixy Pet is getting tired. Feed it some fresh batteries.