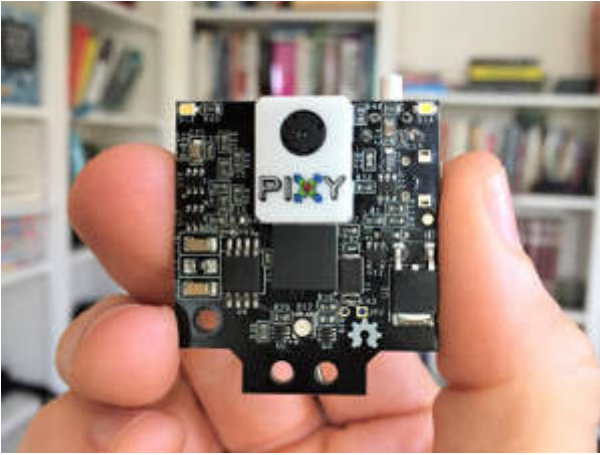


Pixy2 Overview



Pixy2 is the second version of Pixy. It's faster, smaller and more capable than the original Pixy, adding line tracking/following algorithms as well as other features. Here's what we've added to Pixy2:

- Pixy2 detects lines, intersections and small barcodes, intended for line-following robots
- Improved framerate – 60 frames-per-second
- Tracking algorithms have been added to color-based object detection
- Improved and simplified libraries for Arduino, Raspberry Pi and other controllers
- Integrated light source

And of course, Pixy2 does everything that the original Pixy can do:

- Small, fast, easy-to-use, low-cost, readily-available vision system
- Learns to detect objects that you teach it
- Connects to Arduino with included cable. Also works with Raspberry Pi, BeagleBone and similar controllers
- All libraries for Arduino, Raspberry Pi, etc. are provided
- C/C++ and Python are supported
- Communicates via one of several interfaces: SPI, I2C, UART, USB or analog/digital output
- Configuration utility runs on Windows, MacOS and Linux
- All software/firmware is open-source GNU-licensed
- All hardware documentation including schematics, bill of materials, PCB layout, etc. are provided

How Pixy got started

Pixy (CMUcam5) is a partnership between the Carnegie Mellon Robotics Institute and Charmed Labs. Pixy comes from a long line of CMUcams, but Pixy got its real start as a [Kickstarter campaign](#). It first started shipping in March of 2014 and has since become the most popular vision system in history! Pixy is funded exclusively through sales, so thank you for helping make Pixy a success! You can watch the original Kickstarter video below — it's a good introduction!

Vision as a Sensor

If you want your robot to perform a task such as picking up an object, chasing a ball, locating a charging station, etc., and you want a single sensor to help accomplish all of these tasks, then **vision** is your sensor. Vision (image) sensors are useful because they are so flexible. With the right algorithm, an image sensor can sense or detect practically anything. But there are two drawbacks with image sensors: 1) they output lots of data, dozens of megabytes per second, and 2) processing this amount of data can overwhelm many processors. And if the processor can keep up with the data, much of its processing power won't be available for other tasks.

Pixy2 addresses these problems by pairing a powerful dedicated processor with the image sensor. Pixy2 processes images from the image sensor and only sends the useful information (e.g. purple dinosaur detected at x=54, y=103) to your microcontroller. And it does this at frame rate (60 Hz). The information is available through one of several interfaces: UART serial, SPI, I2C, USB, or digital/analog output. So your Arduino or other microcontroller can talk easily with Pixy2 and still have plenty of CPU available for other tasks.

It's possible to hook up multiple Pixys to your microcontroller — for example, a robot with 4 Pixy2's and omnidirectional sensing. Or use Pixy2 without a microcontroller and use the digital or analog outputs to trigger events, switches, servos, etc.

Controller support

Pixy2 can easily connect to lots of different controllers because it supports several interface options (UART serial, SPI, I2C, USB, or digital/analog output), but Pixy began its life talking to Arduinos. We've added support for Arduino Due, Raspberry Pi and BeagleBone Black. Software libraries are provided for all of these platforms so you can get up and running quickly. Additionally, we've added a Python API if you're using a Linux-based controller (e.g. Raspberry Pi, BeagleBone).

60 frames per second

What does "60 frames per second" mean? In short, it means Pixy2 is fast. Pixy2 processes an entire image frame every 1/60th of a second (16.7 milliseconds). This means that you get a complete update of all detected objects' positions every 16.7 ms. At this rate, tracking the path of falling/bouncing ball is possible. (A ball traveling at 40 mph moves less than a foot in 16.7 ms.) If your robot is performing line following, your robot will typically move a small fraction of an inch between frames.

Color Connected Components

Purple dinosaurs (and other things)

Pixy2 uses a color-based filtering algorithm to detect objects called the *Color Connected Components* (CCC) algorithm. Color-based filtering methods are popular because they are fast, efficient, and relatively robust. Most of us are familiar with RGB (red, green, and blue) to represent colors. Pixy2 calculates the color (hue) and saturation of each RGB pixel from the image sensor and uses these as the primary filtering parameters. The hue of an object remains largely unchanged with changes in lighting and exposure. Changes in lighting and exposure can have a frustrating effect on color filtering algorithms, causing them to break. Pixy2's filtering algorithm is robust when it comes to lighting and exposure changes.

Seven color signatures

Pixy2's CCC algorithm remembers up to 7 different color signatures, which means that if you have 7 different objects with unique colors, Pixy2's color filtering algorithm will have no problem identifying them. If you need more than seven, you can use color codes (see below).

Hundreds of objects

Pixy2 can find literally hundreds of objects at a time. It uses a connected components algorithm to determine where one object begins and another ends. Pixy2 then compiles the sizes and locations of each object and reports them through one of its interfaces (e.g. SPI).

Teach it the objects you're interested in

Pixy2 is unique because you can physically teach it what you are interested in sensing. Purple dinosaur? Place the dinosaur in front of Pixy2 and press the button. Orange ball? Place the ball in front of Pixy2 and press the button. It's easy, and it's fast.

More specifically, you teach Pixy2 by holding the object in front of its lens while holding down the button located on top. While doing this, the RGB LED under the lens provides feedback regarding which object it is looking at directly. For example, the LED turns orange when an orange ball is placed directly in front of Pixy2. Release the button and Pixy2 generates a statistical model of the colors contained in the object and stores them in flash. It will then use this statistical model to find objects with similar color signatures in its frame from then on.

Pixy2 can learn seven color signatures, numbered 1-7. Color signature 1 is the default signature. To teach Pixy2 the other signatures (2-7) requires a simple button pressing sequence.

Pixy2 "tracks" each object it detects

Once Pixy2 detects a new object, it will add it to a table of objects that it is currently tracking and assign it a tracking index. It will then attempt to find the object (and every object in the table) in the next frame by finding its best match. Each tracked object receives an index between 0 and 255 that it will keep until it either leaves Pixy2's field-of-view, or Pixy2 can no longer find the object in subsequent frames (because of occlusion, lack of lighting, etc.)

Tracking is useful when you want your program to keep tabs on a certain instance of an object, even though there may be several other similar objects in the frame.

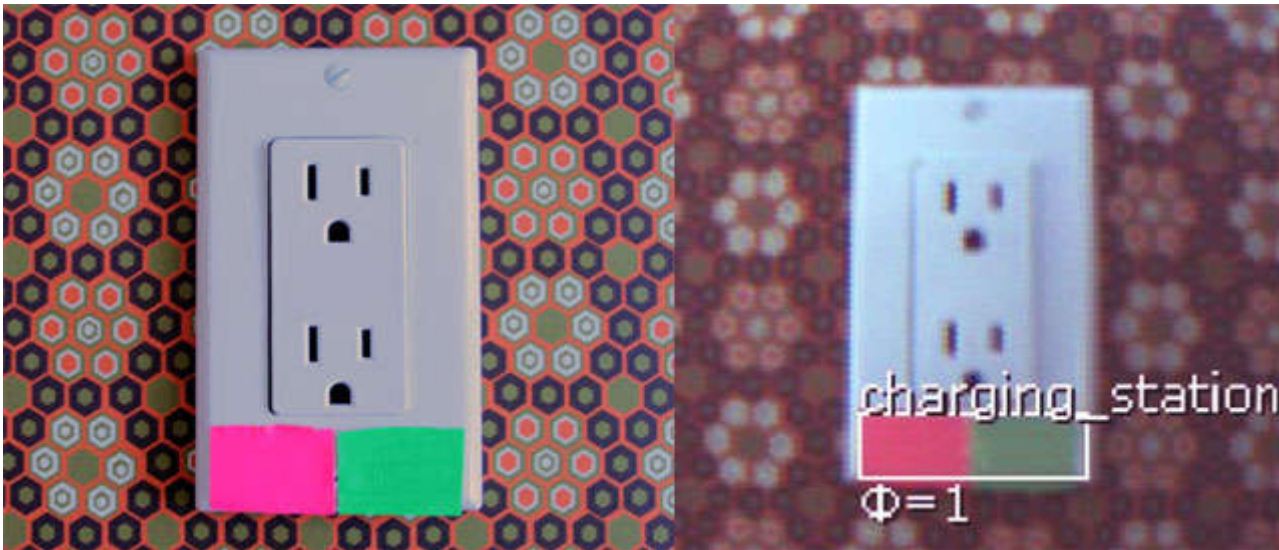
What's a "color code"?

A color code (CC) is two or more color tags placed close together. Pixy2 can detect and decode CCs and present them as special objects. CCs are useful if you have lots of objects you want to detect and identify (i.e. more than could be detected with the seven separate color signatures alone.)

A color code scheme with 2 tags and 4 different colors can differentiate up to 12 unique objects. CCs with 3, 4 and 5 tags and/or more different colors are possible and can allow for many, many more unique objects. (In fact, thousands of unique codes are possible by using CCs with 5 tags and 6 colors.)

Why Color Codes?

CCs are useful if you have lots of objects you want to detect and identify, more than could be detected with the seven separate color signatures alone. CCs also improve detection accuracy by decreasing false detections. That is, there is a low probability that specific colors will occur both in a specific order and close together. The drawback is that you need to place a CC on each object you're interested in detecting. Often the object you're interested in (yellow ball, purple toy) has a unique color signature and CCs aren't needed. Objects with CCs and objects without CCs can be used side-by-side with no problems, so you are free to use CCs for some objects and not others.



CCs give you an accurate angle estimate of the object (in addition to the position and size). This is a computational “freebie” that some applications may find useful. The angle estimate, decoded CCs, regular objects and all of their positions and sizes are provided at 60 frames per second.

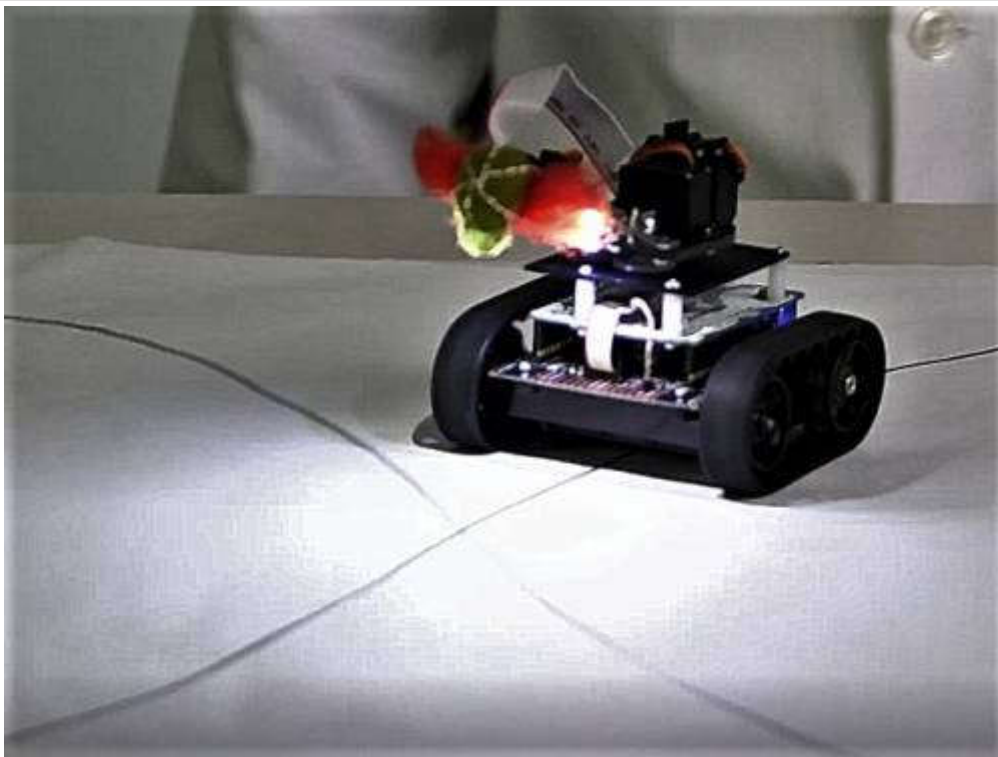
CCs might be particularly useful for helping a robot navigate. For example, an indoor environment with CCs uniquely identifying each doorway and hallway would be both low-cost and robust.

2018/05/20 16:27 · pixycam

Line tracking for line-following

Pixy2 has added the ability to detect and track lines. Line-following is a popular robotics demo/application because it is relatively simple to implement and gives a robot simple navigation abilities. Most line-following robots use discrete photosensors to distinguish between the line and the background. This method can be effective, but it tends to work best with only thick lines, and the sensing is localized making it difficult for the robot to predict the direction of the line or deal with intersections.

Pixy2 attempts to solve the more general problem of line-following by using its image (array) sensor. When driving a car, your eyes take in lots of information about the road, the direction of the road (is there a sharp curve coming up?) and if there is an intersection ahead. This information is important! Similarly, each of Pixy2's camera frames takes in information about the line being followed, its direction, other lines, and any intersections that these lines may form. Pixy2's algorithms take care of the rest. Pixy2 can also read simple barcodes, which can inform your robot what it should do – turn left, turn right, slow down, etc. Pixy2 does all of this at 60 frames-per-second.

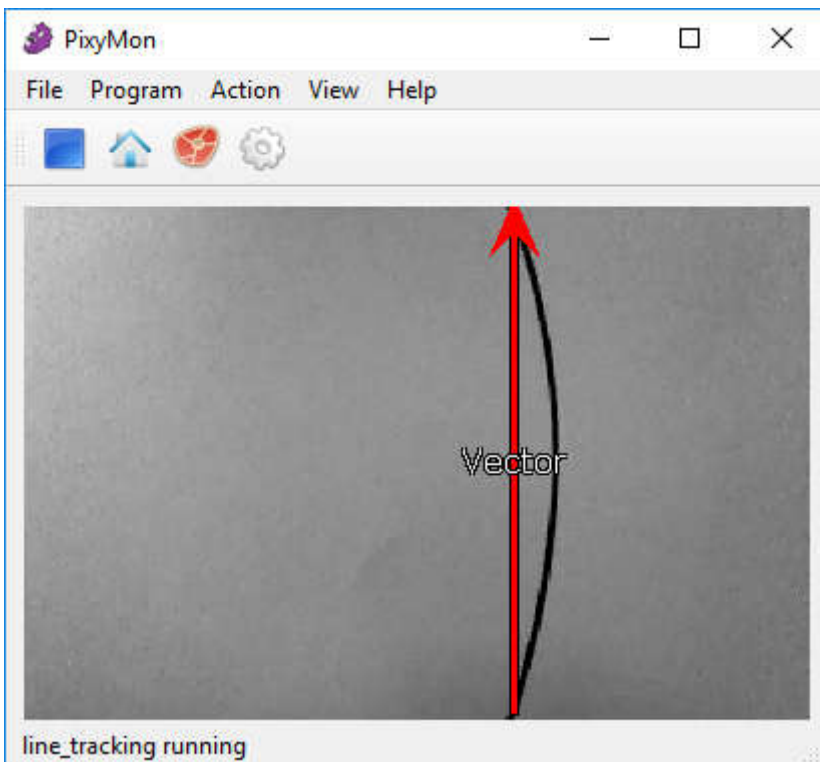
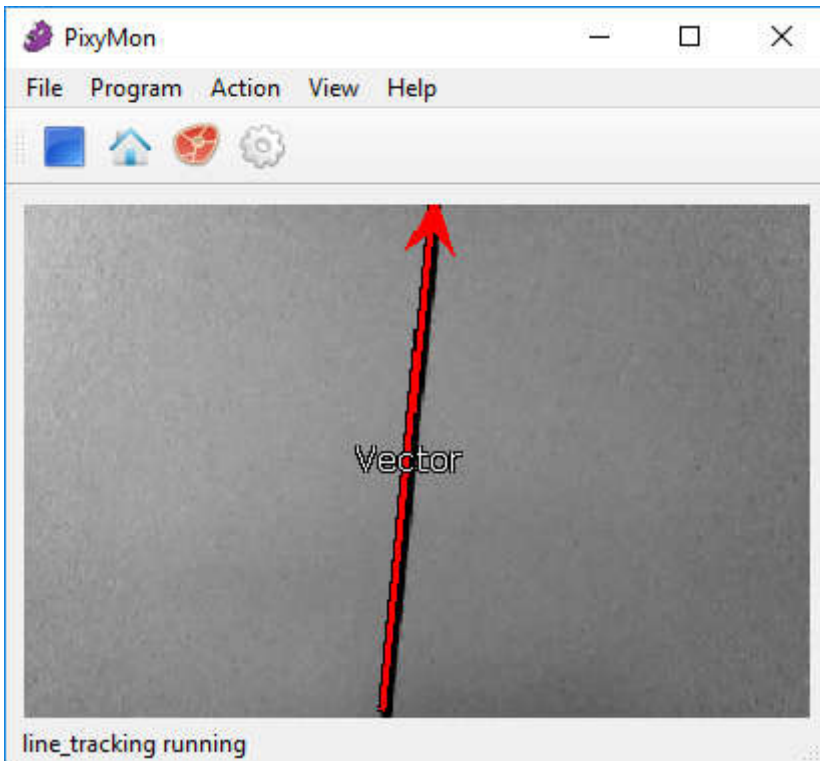


Pixy2 and pan-tilt on top of Zumo base, looking down and following lines, sporting wig, little baseball cap

Detecting and tracking lines

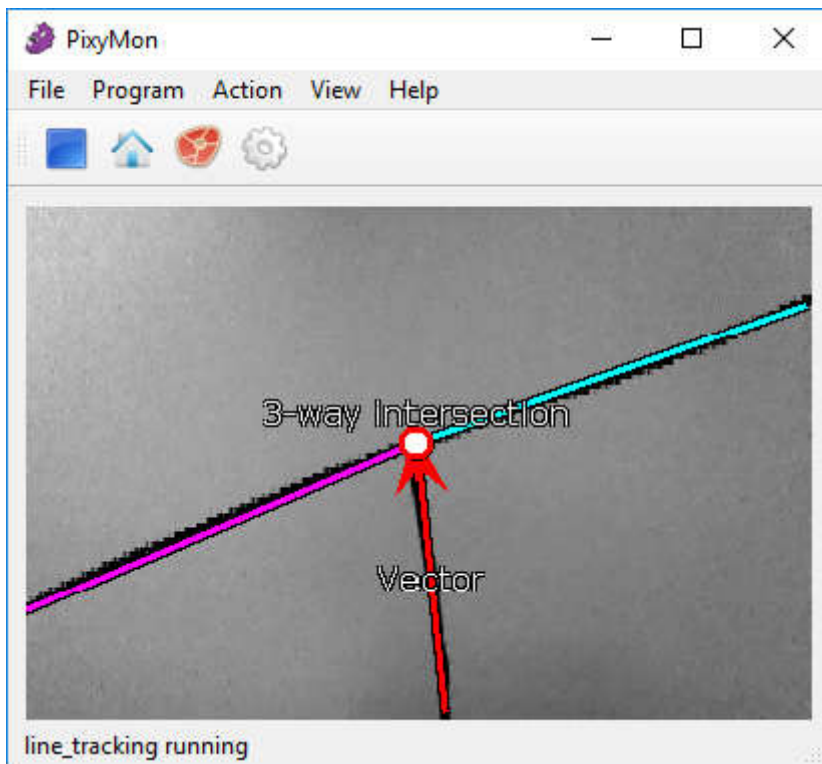
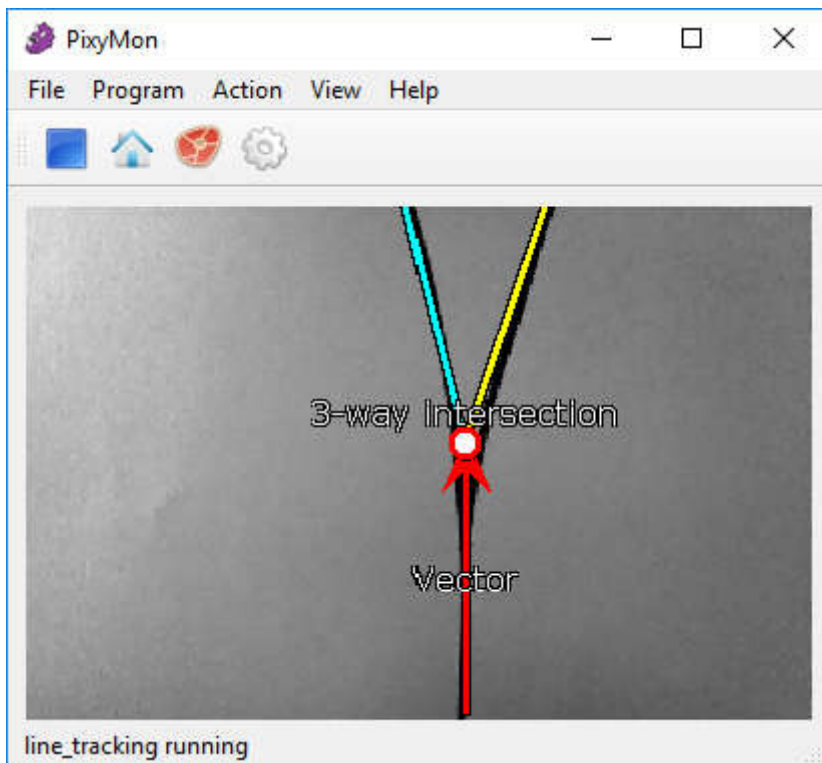
Pixy2's algorithms can detect dark lines on a light background or vice-versa. You can also tell Pixy2 that you are only interested in lines within a certain range of widths. Pixy2 goes through each frame finding lines that meet your criteria and determining where each line begins and ends within the frame.

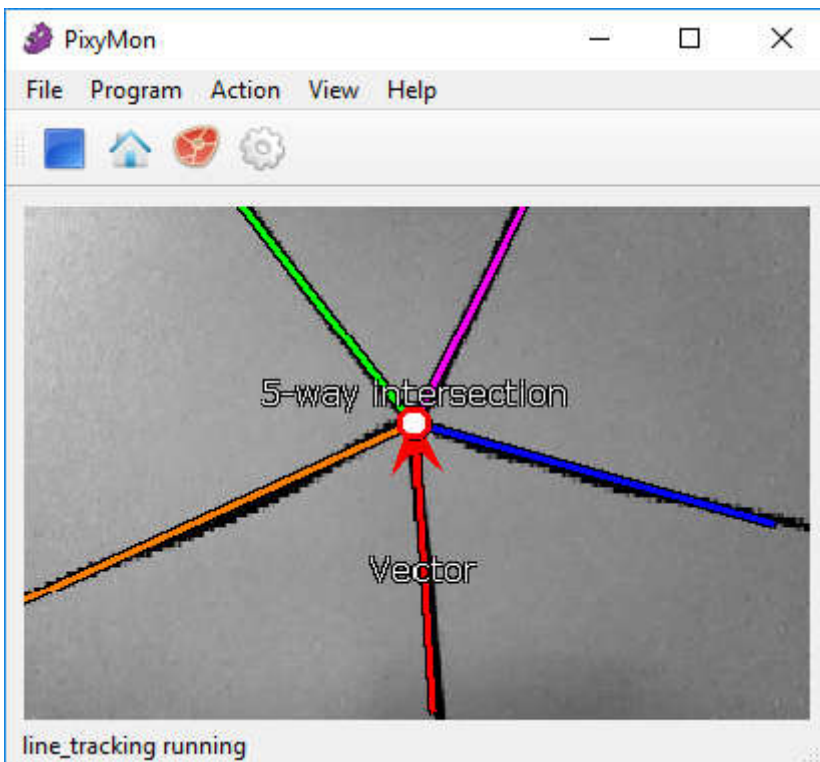
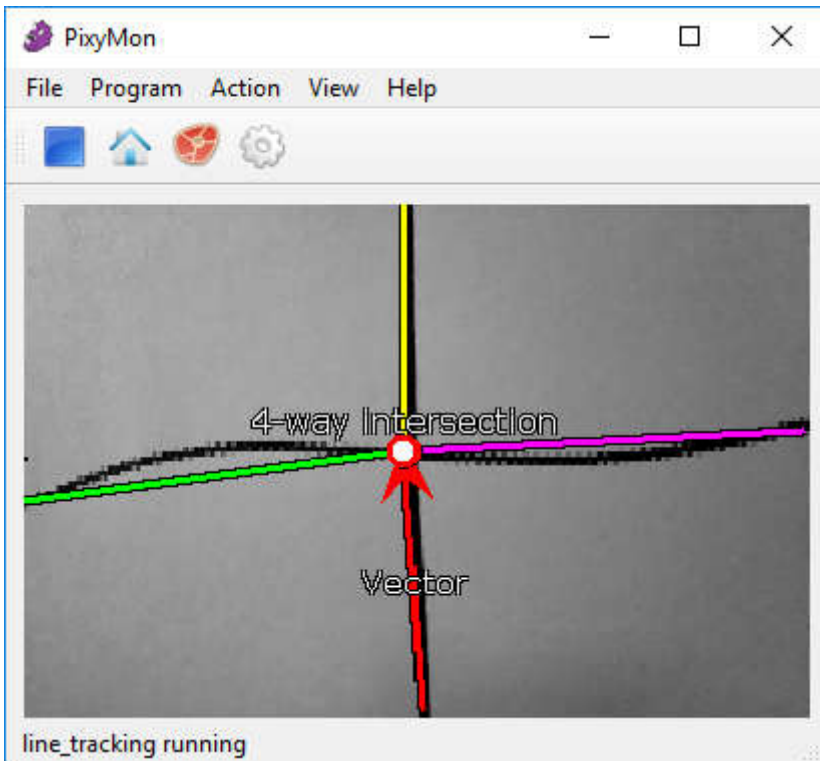
Normally, a robot is only interested in the line that it is currently following. If new lines appear in the frame, you typically don't want your robot to switch and start following those lines. Pixy2 tracks each line, determining where each line moves in subsequent frames, so the line your robot is following remains so unless you tell it otherwise. The line your robot is currently following is called the **Vector**. Pixy tells you where the Vector starts and ends in each frame, so you can use that information to determine your robot's motion. So, for example, if the Vector heads toward the right in the frame, your robot should start turning right. If the Vector heads toward the left, your robot should start turning left, etc. We have example programs that run on an Arduino that show how to do this.



Detecting Intersections and "branching"

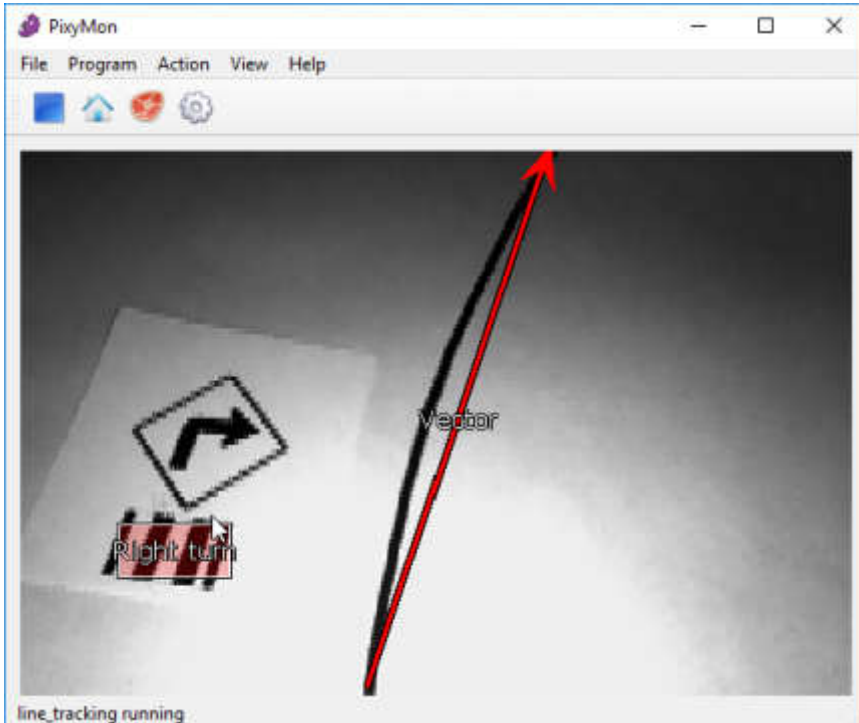
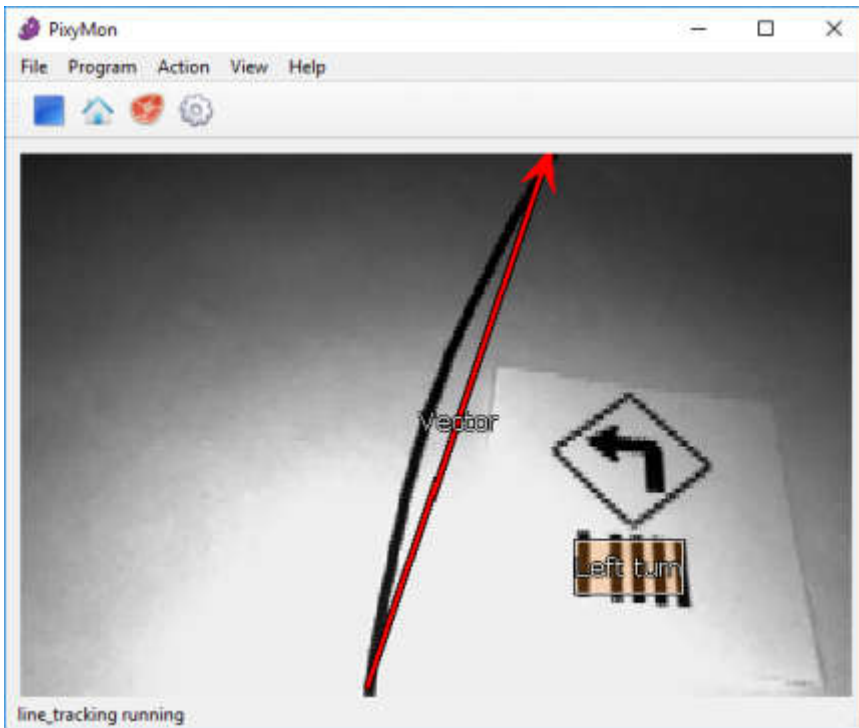
Intersections can be challenging for a line-following robot because they can take so many different shapes and forms. Will your robot deal well with T-shaped intersections, but fail when it sees Y-shaped intersections? What about 4 or even 5-way intersections? What if the intersections and lines are hand-drawn or "curvy"? Pixy2 can handle these cases. For example, Pixy2 tells your program that it has detected an intersection ahead, how many branches are in the intersection and the angles of each branch. Your program can then decide which branch it wants to take and then communicate it to Pixy2. Pixy2 will then make the branch its new Vector line, and your robot will in essence "take the branch" by following the new Vector's direction.





Barcodes

In addition to detecting lines and intersections, Pixy2 can detect small barcodes. The barcodes can be used to tell your program what to do at the next intersection, or provide a hint, such as "slow down". Pixy2 can detect up to 16 different codes and report to your program when it detects each one. Your program can choose how it reacts to each barcode.



Just give me the features

Pixy2 tries to send only the most relevant information to your program:

- It finds the best Vector candidate and begins tracking it from frame to frame. The Vector is often the only feature Pixy2 provides to your program, and it's the only feature your program typically needs when following a line.
- It detects intersections and reports them after they have met the filtering constraint.
- It detects barcodes and reports them after they have met the filtering constraint.
- It reports each new barcode and intersection *one time* so your program doesn't need to keep track of which features it has/hasn't seen previously.
- It "executes" branches by assigning the Vector to the branch line that your program chooses.

But you may want to use Pixy2's line tracking algorithms in a different way that requires that Pixy2 share more of the information that it is gathering. For example, you may want your program (not Pixy) to choose the Vector line. If so, Pixy2 can provide your program with all lines, intersections and barcodes that it detects, regardless of filtering constraints or the state of the Vector. Pixy2 will still track each line, intersection and barcode from frame to frame and provide your program with this information, but your program can choose to use the information or ignore it. In other words, your program will be unconstrained regarding the information it receives from Pixy2.

Integrated light source

Pixy2 has an integrated light source for when the lighting needs a boost. The light source is particularly useful when your robot is performing line-following. It gives Pixy2 plenty of light regardless of the level of ambient light and it reduces motion blur, which can be an issue when your robot is going fast and the ground is zipping by underneath it. The integrated light source emits around 20 lumens, about twice the amount an iPhone in flashlight mode emits.

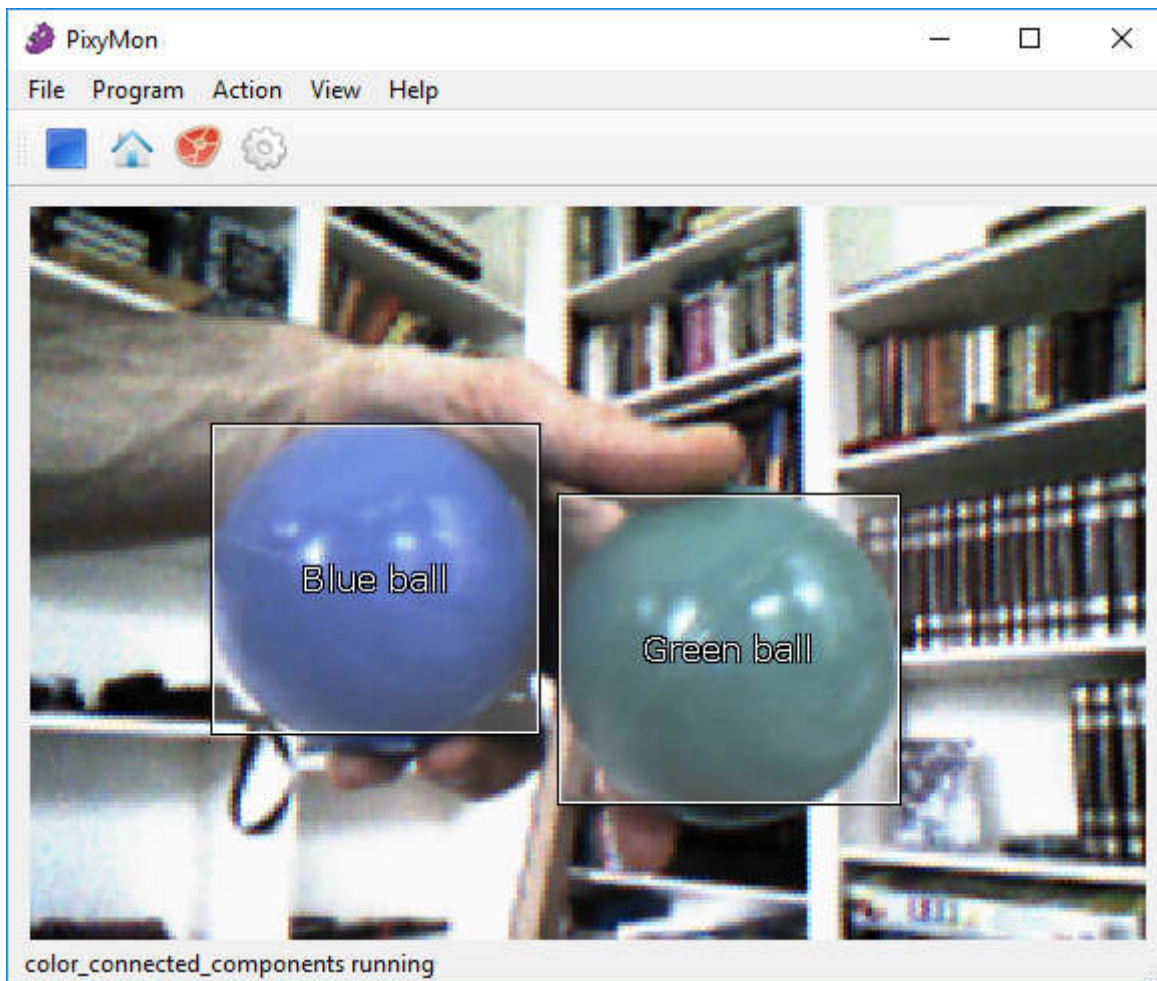


2018/05/21 15:39 · pixycam

PixyMon lets you see what Pixy2 sees

PixyMon is an application that runs on Windows, MacOS and Linux. It allows you to see what Pixy2 sees, either as raw or processed video. It also allows you to configure your Pixy2, set the output port and manage color signatures. PixyMon communicates with Pixy2 over a standard mini USB cable.

PixyMon is great for debugging your application. You can plug a USB cable into the back of Pixy2 and run PixyMon and then see what Pixy2 sees while it is hooked to your Arduino or other microcontroller — no need to unplug anything. PixyMon is open source, like everything else.



Technical specs

- Processor: NXP LPC4330, 204 MHz, dual core
- Image sensor: Aptina MT9M114, 1296×976 resolution with integrated image flow processor
- Lens field-of-view: 60 degrees horizontal, 40 degrees vertical
- Power consumption: 140 mA typical
- Power input: USB input (5V) or unregulated input (6V to 10V)
- RAM: 264K bytes
- Flash: 2M bytes
- Available data outputs: UART serial, SPI, I2C, USB, digital, analog
- Dimensions: 1.5" x 1.65" x 0.6"
- Weight: 10 grams
- Integrated light source, approximately 20 lumens

