What Does it Mean to be Autonomous?

Basics of Autonomy

- Action (Actuators)
  - Locomotion: Wheels (Differential Drives, Kinematics)
- Perception (Sensors)
  - Proprioception and Exteroception (Bumper & Depth)
  - Today: RGB Cameras and Video
- Cognition (Control)
  - Reactive Behaviors (e.g. Roomba & Wanderer)
  - Today: PID Controllers
Feedback Control and PID

Closed Loop Control
- Desired state (goal, setpoint)
- Feedback (i.e., measured - desired)
- Goal: MAINTAIN set-point

Example: Thermostat
- World (Room)
- Sensor (Temperature)
- Actuation (Heater)
- Controller (Thermostat)

Desired goal

Example: Wall Following Robot
- Simple scenario: trying to move along an infinite straight wall while maintaining a fixed distance.

Generic Program Loop
- Move 1 step forward
- If distance-to-wall > desired,
  Then turn towards the wall
  Else turn away from the wall

Concrete Program Loop
- Move 0.5 body-length forward
- If distance-to-wall > desired,
  Then turn 45 degrees towards the wall
  Else turn 45 degrees away from the wall

How does this Program perform?
Example: Wall Following Robot

- Simple scenario: trying to move along an infinite straight wall while maintaining a fixed distance.
- Concrete Program Loop
  - Move 0.5 body-length forward
  - If distance-to-wall > desired,
    - Then turn 45 degrees towards the wall
    - Else turn 45 degrees away from the wall
- How does this Program perform?
- How do we do better?

How does this Program perform?
- Oscillates!!

How do we do better?
- Reduce turning angle to be very small (avoid overshoot)
- Check for error very frequently (avoid overshoot)
- Define some “slop” in our goal (range instead of exact)
- Sometimes “bang-bang” control is enough (e.g. roomba using bump sensors to wall-follow)

How do we do even better? Use more information!

Proportional (P) Control

- Use more information: use both the direction and magnitude of the error to decide how to adjust.
- Error = distance-to-wall – desired distance
- Adjustment
  - \( \text{ChangeAngle} = K_p \times \text{error} \)
  - Current action is just your past action + adjustment
  - \( K_p = \text{“gain”} \)

High-level idea: adjust proportional to the error
- If far from the Dline — we will turn sharply
- If we are close to the Dline — then turn very slowly
- How do we decide what \( K_p \) is?
- Model or Experiments (Control Theory)
**Proportional (P) Control**

Proportional Control Program Loop
- Move 0.5 body-length forward
- If distance-to-wall > desired,
  - let error = |desired – distance-to-wall|
  - Then turn $K_p \times error$ towards the wall
  - Else turn $K_p \times error$ away from the wall

**P-controllers are very useful!**

**New Scenario**
Orient towards a "Source"

**P-controllers are very useful!**

Wall Following | Visual Homing | Centering | Collision Avoidance
---|---|---|---

Reactive Behaviors == Feedback Controllers

**When P Control is not enough**

**P Controller**
- Loop
- Measure error
- Apply Accelerator = $K_p \times error$
- as I get closer, I apply less gas

**Ignores inertia!**
- Momentum = mass*velocity
- Car (heavy) at 10mph vs 100mph
- P-control only reacts to current "error"
- But error is changing also based on speed
- Can we "predict the future change in error"?

**PD Controller**
- Loop
- Measure error = distance-to-wall
- Derivative-error = $d(error)/dt$
- Change = $K_p \times error - K_d \times deriv_error$

**Hint:** Pset 3 "Follower"
When P Control is not enough

What if there is an "external" constant source of error?

P Controller
Loop
Measure error
Apply Accelerator = k \cdot error

Adjust based on "past failures"

PID Controller
Loop
Measure error = distance-to-wall
Derivative-error = d(error)/dt
Integral-error = sum(error + past)
Change = K_p \cdot error - K_d \cdot deriv-error + K_i \cdot integral-error

And that’s PID Control!

Proportional Integral Derivative

u(t) = K_p e(t) + K_d \frac{d}{dt} e(t) + K_i \int e(t)

P-control: In this class, we will only really use P-controllers since our robots are slow. Derivative control while important is the most complex, since derivatives tend to be noisy. Integral control is more commonly used, to get rid of persistent errors.

Setting Gains: Analytical models are hard to get accurate, but empirical tuning is often not that bad. Common method is to tune K_p first, until stable consistent oscillations, then tune K_d and then K_i. There is also a heuristic method called the Ziegler-Nichols Tuning Method which defines the desirable K_p:K_d:K_i ratio.

Basics of Autonomy

**ACTION**
- Action (Actuators)
- Locomotion: Wheels (Differential Drives, Kinematics)

**PERCEPTION**
- Perception (Sensors)
- Proprioception and Exteroception (Bump, Depth)
- Today: More about Cameras and Color

**COGNITION**
- Cognition (Control)
- Reactive Behaviors (e.g. roomba, collision avoidance)
- Today: PID Controllers

Perception: Robot Vision

- Why Robot Vision?
- Operate in human designed world!
- Cheaper and cheaper Cameras!
- But robots vision \neq computer vision
- Robots have limited computation time and not a lot of memory (real-time)
- Robots are action driven, and thus perception is task driven – can be less general (minimization)
- Robots also have the advantage (?) that they see images over and over while they move (video)
**Vision: Many Options**

- **COLOR CAMERAS**
  - Object Recognition
  - Classically hard AI problem!
  - Camera gives an array of light pixels
  - How do you recognize a chair?

- **DEPTH SENSING**
  - Task-driven approach
  - Segmentation (shape/color characteristics)
    - Colorspaces (HSV)
    - Typical Styles: Blur => Mask => Contours [LAB 2]
    - OpenCV (real-time vision)
  - Non-Segmentation ("features")
    - Template Matching and Histogram Backprojection
    - Classifiers ("Face Detection")
    - Fiducials (e.g. AprilTag)

- **VIDEO/MOTION**
  - Why Object Recognition is hard.
  - How Depth Cameras work & When to use
  - Optic Flow and Object Tracking

---

**But First: A Video......**

**Vision: Many Options**

- **COLOR CAMERAS**

---

**Segmentation: Color Space**

- Digital Camera = Array of pixels (pixel == “picture element”)
  - **RGB**
    - 24 bit (0-255, 0-255, 0-255)
  - **HSV or HSI**
    - Hue = actual color
    - Saturation = amount of color
    - Intensity = amount of light

Equivalent to RGB, but easier to numerically threshold on human “meaningful” notions of color

---

- I can easily identify a RED object even if its dark or sunlight is on it!
Segmentation: Blur

- Gaussian Filter (one of many possibilities)

Noisy pixels in the image are removed.

Use a Filter to "smooth" the image out.

Filtering is a general concept: apply a matrix to every pixel.

Segmentation: Blur => Mask

- MASK == Threshold image based on "Color"
  - Can combine masks
  - Can “Posterize” (assign color bins)

Segmentation: Blur => Mask => "Blob"

- Give me Objects!
  - Segment my image into "contiguous regions" of color (blob)
  - OpenCV: Find Contours — gives you a curve around each object
    (curve is represented by an array of boundary points)
  - Then you can do stuff! (bounding box, areas)

Digression: Find Blobs Algorithm

- Run-Length Encoding:
  - Find the contiguous row-regions of color of choice
  - For each row
    - While there are still pixels in the row
      - Discard pixels until see red
        - Record start of a "run" by (row, column)
      - Discard pixels until see black
        - Record end of a "run" by (row, column)
  - Region Extraction
    - Link together the row-runs that touch in columns
    - Create a directed graph over the row-runs

Run: [(2,4) to (2,6)] [(2,12) to (2,15)]
[(3,4) to (3,6)] [(3,12) to (3,15)]
[(5,4) to (5,15)]
[(8,4) to (8,6)] [(8,12) to (8,15)]
**Segmentation: Blur => Mask => “Blob”**

- **OpenCV libraries** make much of this very easy
  - Good documentation and online examples
  - *BUT still need lots of testing! (customize to your errors)*

- **Lab2 Solutions repository** has lots of goodies
  - Example of blur=>mask=>contours
  - Trackbar! For calibrating HSV bounds

---

**Segmentation to Object Size**

If you know the real object size, then the image tells you how far it is!

But even better approach is to combine RGB Camera and Depth Camera images.

---

**Vision: Many Options**

**COLOR CAMERAS**

- **Object Recognition**
  - Classically hard AI problem!
  - Camera gives an array of light pixels
  - How do you recognize a chair?

  **Task-driven Approach**
  - **Segmentation** (shape/color characteristics)
    - Colorspaces (HSV)
    - Typical style: Blur => Mask => Contours
    - Opencv (real-time vision)
  - **Non-Segmentation (“features”)**
    - Template Matching and Histogram Backprojection
    - Classifiers ("Face Detection")
    - Fiducials (e.g. AprilTag)

---

**Non-Segmentation Approaches**

You don’t need to always “recognize” the objects in your image – as the background gets more cluttered and complex this becomes hard anyways....

- **Image Signature**
  - Template matching (“image” itself)
  - Color Histogramming (pixel distribution)
  - Classifiers (requires training data)
  - Cascade Classifiers (face detection)
Image Signatures

**Template**
- Take a closeup of the "desired" object

**Match**
- Image Region – Template (sliding window, rotate template)

Output is an image of values where the lowest value is the best match.
Scale/size invariance requires doing this many times.

Problem: too detail oriented

*OpenCV: see Template Matching, Histogram Backprojection, and Image Pyramids*

**Signature**
- Take a closeup of the "desired" object

**Match**
- Histogram(Image Region) – signature

Example: Robot "imprints" on an object. Then robot would move with a speed proportional to the match… Follows a purple triangle too…..

Instead of Color you can use More Robust Features
- Edge detection: (Sobel or Canny)
- Corner Detection: (Harris)

SIFT = Scale Invariant Feature Detection
Do this process at many scales and rotations

Non-Segmentation Approaches

You don’t need to always “recognize” the objects in your image – as the background gets more cluttered and complex this becomes hard anyways.....

- **Image Signature**
  - Template matching (“image” itself)
  - Color Histogramming (pixel distribution)
  - Classifiers (Requires training data)
  - Cascade Classifiers (e.g. Face Detection)

Nothing is perfect!

Fiducials
- Place easy to recognize landmarks in your environment

*AprilTag System, Ed Olson, Univ of Michigan*
**Outline**

COLOR CAMERAS  DEPTH SENSING  VIDEO/MOTION

Object Recognition
(segmentation vs non-segmentation)  How Depth Cameras work & When to use

**Video!**

Motion can reveal many things!
- Background subtraction (humans move!)
- Optic flow (recover motion)
- Tracking objects

[Compare frames in RGB or Depth!]

*These slides are adapted from OpenCV tutorial (which is great reading! docs.opencv.org)
And OpenCV provides implementations that you can use out of the box.

**Basic idea** – take a “window” of frames and look at all the pixels that don’t change (or median pixel value). Subtract from your image...

Smarter algorithms: GMM and Bayesian models of the background

**Video!**

Motion can reveal many things!
- Background subtraction (humans move!)
- Optic flow (recover motion)
- Tracking objects

[Compare frames in RGB or Depth!]

Instead of just subtraction, try to “track” where each pixel moved to.

Can give you SPEED and DIRECTION! And segmentation...

Color represents direction. Brightness represents speed.
Optic Flow

Robotics! Can recover your own motion
- Speed (magnitude and direction of arrows)
- Or Distance to objects (at given speed)

Can also recover “Behavior”

These algorithms depend on “feature matching”
- Pixel window matching (dense OF) or track features (corners/sift)

Video!

Motion can reveal many things!
- Background subtraction (humans move!)
- Optic flow (recover motion)
- Tracking objects

[Compare frames in RGB or Depth!]

Basic idea: Follow a “subwindow” as it moves through the image.

OpenCV: Combine Histogram Backprojection + Camshift
Later: Kalman Filter

Digression: Kalman Filter

Motion Model Prediction (T+1)
Image Model Prediction (T+1)
Both provide independent Probability(pix)
Compute normalized sum
Get a confidence value for your tracked object!

Outline

COLOR CAMERAS
- Object Recognition
  (segmentation vs non-segmentation)

DEPTH SENSING
- How Depth Cameras work & When to use

VIDEO/MOTION
- Optic Flow and Object Tracking
Vision is Complex

- We still understand very little about human visual cortex
- Much less than the eye “hardware”
- We do understand that animal vision systems use tricks
  - Bees, spiders, fish, employ many tricks that are Task Specific
  - And just good enough - not “logical” or fool proof.
- For Robots, finding appropriate tricks is critical
  - Not just for simple robots like Turtlebot
  - Google Self-Driving Car (“background substraction”)
- Finally Vision is just one sensor out of many sensors we have;
  Choose the right sensor for the job
  - Human existence does not rely on vision – touch, balance, sound

Upcoming: Pset 3 Follower

- You have a GREEN band to put on your ankle
- Part 3(a) Your robot should recognize the band
  - Draw a bounding box around the ankle band
  - Try to recognize at least up to 4 feet away
  - Calibrate! (“trackbar”)
- Part 3(b) Your robot should follow the band
  - P-control will be helpful to adjust quickly
  - Avoid running into obstacles
  - Will need to deal with quick disappearance (other leg blocks it) vs longer disappearance (robot lost you)