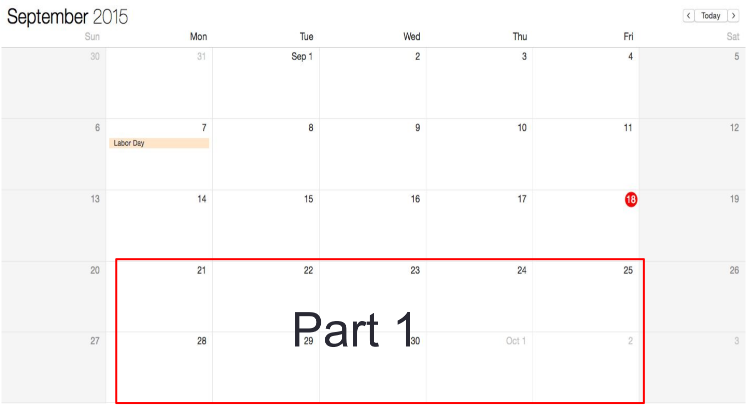
CS123 - Recap2 & Final Project

Programming Your Personal Robot

Kyong-Sok "KC" Chang, David Zhu
Fall 2015-16

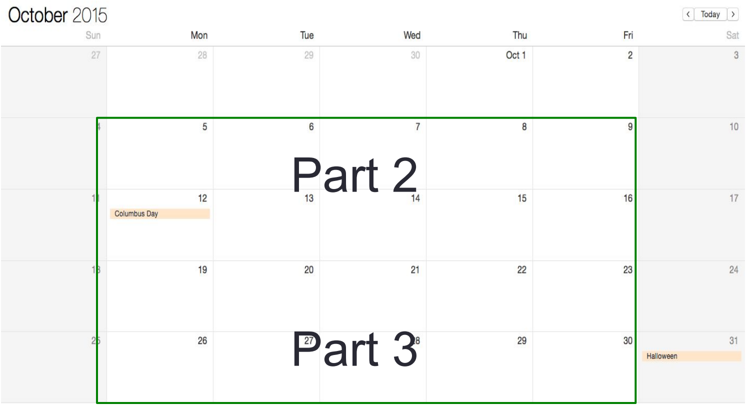
Calendar



September 2015

Sun	Mon	Tue	Wed	Thu	Fri	Sat
30	31	Sep 1	2	3	4	5
6	7 Labor Day	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	Oct 1	2	3

Part 1

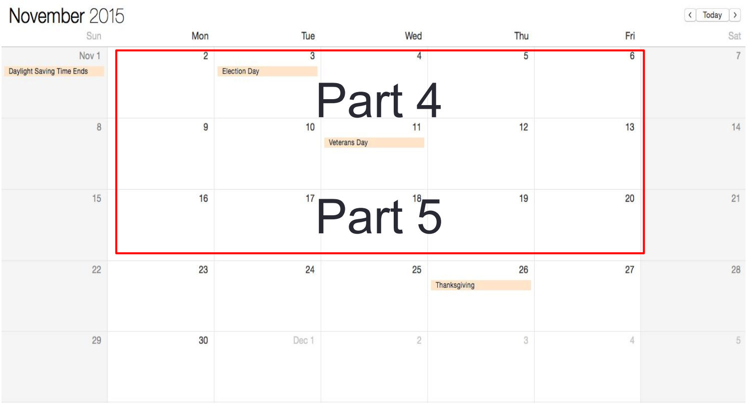


October 2015

Sun	Mon	Tue	Wed	Thu	Fri	Sat
27	28	29	30	Oct 1	2	3
4	5	6	7	8	9	10
11	12 Columbus Day	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31 Halloween

Part 2

Part 3

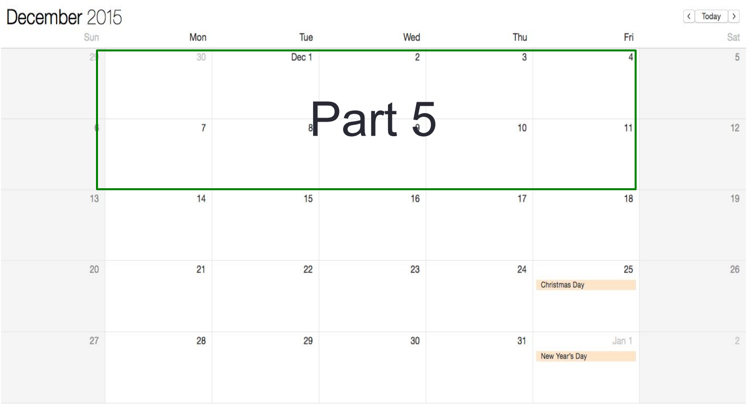


November 2015

Sun	Mon	Tue	Wed	Thu	Fri	Sat
Nov 1 Daylight Saving Time Ends	2	3 Election Day	4	5	6	7
8	9	10	11 Veterans Day	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26 Thanksgiving	27	28
29	30	Dec 1	2	3	4	5

Part 4

Part 5



December 2015

Sun	Mon	Tue	Wed	Thu	Fri	Sat
29	30	Dec 1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25 Christmas Day	26
27	28	29	30	31	Jan 1 New Year's Day	2

Part 5

KC
Teaching

David
Teaching

Syllabus

- Part 1 - Communicating with robot (2 weeks)
  - BLE communication and robot API
- Part 2 - Event Driven Behavior (2 weeks)
  - Finite State Machine (Behavior Tree)
- Part 3 - Reasoning with Uncertainty (2 weeks)
  - Dealing with noisy data, uncertainty in sensing and control
- Part 4 - Extending the robot (1 weeks)
  - I/O extensions: digital, analog, servo, pwm, etc
- Part 5 – Putting it together (including UI/UX) (3 weeks)
  - Design and implement of final (group) project
  - Encourage you to go “above and beyond”

Logistics

- TA sessions (office hours): this week
  - Location: Gates B21 (Th: Huang basement)
  - Time: M:2~4pm, Tu:2~4pm, W:12:30-2:30pm, Th:2~4pm
- Lab reserved for CS123: this week
  - MTuW: 12~6pm @ Gates B21
- My office hours (KC)
  - Tues: 1-2pm @ Gates B21(Tu)
-

Robotics Company: New vs. Old

- Apple, Samsung
- Tesla, LG
- Google, Alibaba, Naver
- Softbank, SKT
- Foxconn
- Toyota, Honda
- Amazon
- Disney
- iRobot, reThink
- Aethon, Savioke, Fetch
- Yujin, SimLab, Wonik

- ABB
- Fanuc
- Yaskawa
- Adept
- Denso
- Kawasaki
- Kuka
- Mitsubishi
- Schunk
- Staubli
- Yamaha

Outline

- Logistics
- Future robots: New Robotics Company
- Recap: Part 1~4 (more on Part 2 and 3)
- Part 5: Putting it together (Navigation)
- Final projects
  - Mobile Robot Programming
    - Event-driven programming: FSM
    - Modeling
    - Localization
    - Planning
    - Execution
    - UI / UX
    - Creative

Objectives

- Expose to the challenges of robot programming
  - Gain a better understanding of the difficulty of programming in the real (physical) world
  - Appreciate the challenges of programming in the real world
- Learn basic concepts and techniques
  - Event driven programming: FSM
  - Modeling the robot: mapping b/w Real world and Virtual world
  - Localization & Planning & Execution
- "Opened" problems
  - No 100% guaranteed solution
  - You can always do better
- "Not well defined" problems
  - Further constraining and decompose the problem

Lec#05: Event Driven Behavior

- 2.1 Event Driven Programming
  - Programming Paradigms and Paradigm Shift
  - Event Driven Programming Concept
    - Tkinter – as a simple example
  - More on threads
  - Implementation of a simple event driven behavior for Hamster
- 2.2 Finite State Machine
  - Concept of FSM
  - Implementation details (a simple FSM for Hamster)
  - FSM driven by an event queue
- 2.3 Related Topics and Discussion
  - Concept of HFSM and BT

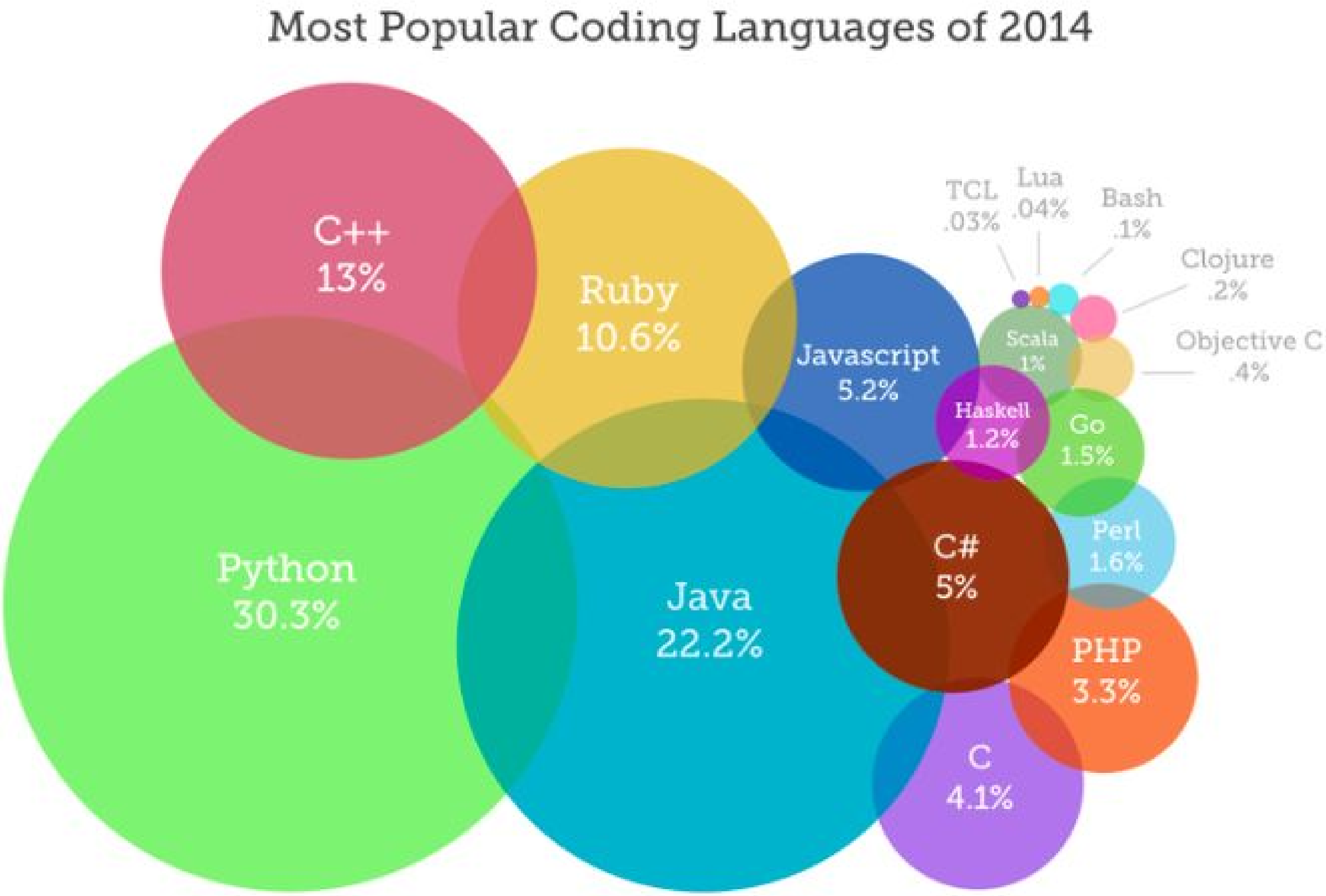
(if time allows, not needed for projects)

Comparing Different Paradigms

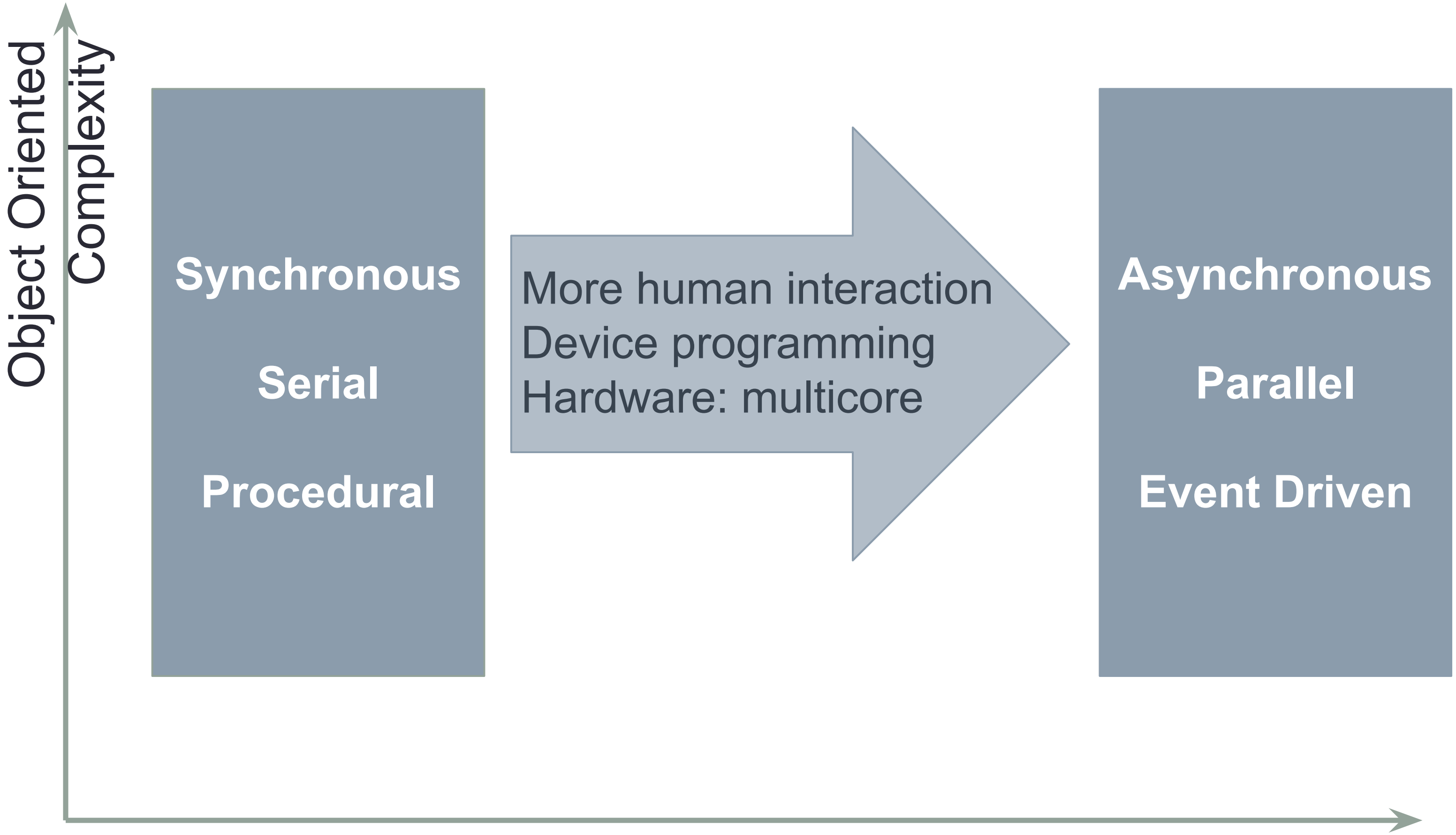
Different “axis” to organize/compare these paradigms

- Declarative vs. Imperative
  - What you want vs. How to do it
- Procedural vs. Event-Driven
  - Step-by-step vs. Event driven

Programming Languages



Programming Paradigm “Shift”



Synchronous

Serial

Procedural

More human interaction
Device programming
Hardware: multicore

Asynchronous

Parallel

Event Driven

Choosing A Paradigm: What to Consider?

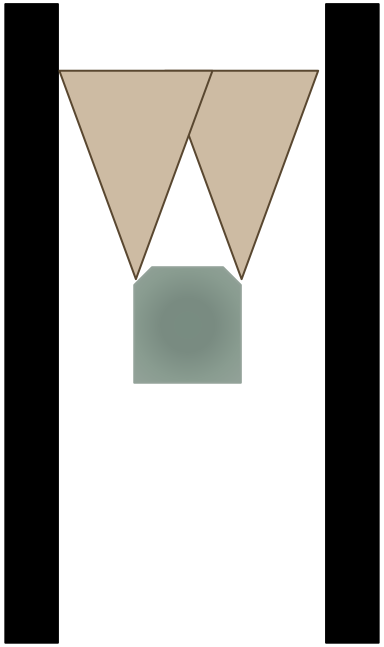
- Suitable for problem formulation
- Ease of implementation
  - clarity
  - debugging
- Scalability
- Efficiency

How to Characterize Robot Programming

How to Characterize Robot Programming

- Open-loop Control
  - Execute robot actions without feedbacks
- Closed-loop Control
  - Adjust robot actions (motion) base on sensor feedbacks, thus compensate for errors

Closed-loop Control



Hallway following

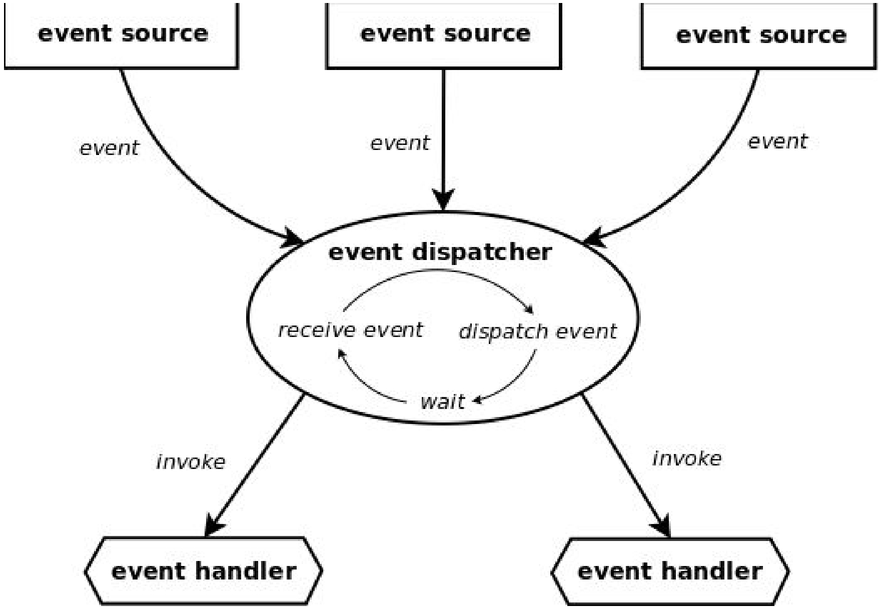
- Adjust robot actions (motion) base on sensor feedbacks, thus compensate for errors
- Necessary because of incomplete and imperfect model of the world, and because of control uncertainty

Event Driven Programming

- Event Driven (Event-based) Programming is a programming paradigm is which the flow of the program is determined by events
- Common examples:
  - Games
  - Web UI
  - Robot

Event Driven Programming

- Event Dispatcher
  - Monitor events and "dispatch" to handlers
- Event Handlers
  - Program waits for events
  - When certain events happen, the program responds and does something (or decides to do nothing)



Lec#06: Event Driven Behavior 2

- Threads
  - What are threads?
  - Why use threads?
  - Communication between threads?
- Queues
  - FIFO vs. Priority
  - Multi-thread safe
- Implementing an Event System using Threads and Queue
  - Dispatcher
  - Handlers
- Folder Structure (Behavior Package)
- Assignment#2-1: Escape

What are Threads

Running several threads is similar to running several different programs concurrently, but with the following benefits:

- Multiple threads within a process share the same data space with the main thread and can therefore share information or communicate with each other more easily than if they were separate processes.
- Threads sometimes called light-weight processes and they do not require much memory overhead; they are “cheaper” than processes.

What are Threads For?

- Threads are used in cases where the execution of a task involves some waiting
- So we can execute multiple tasks “at the same time”

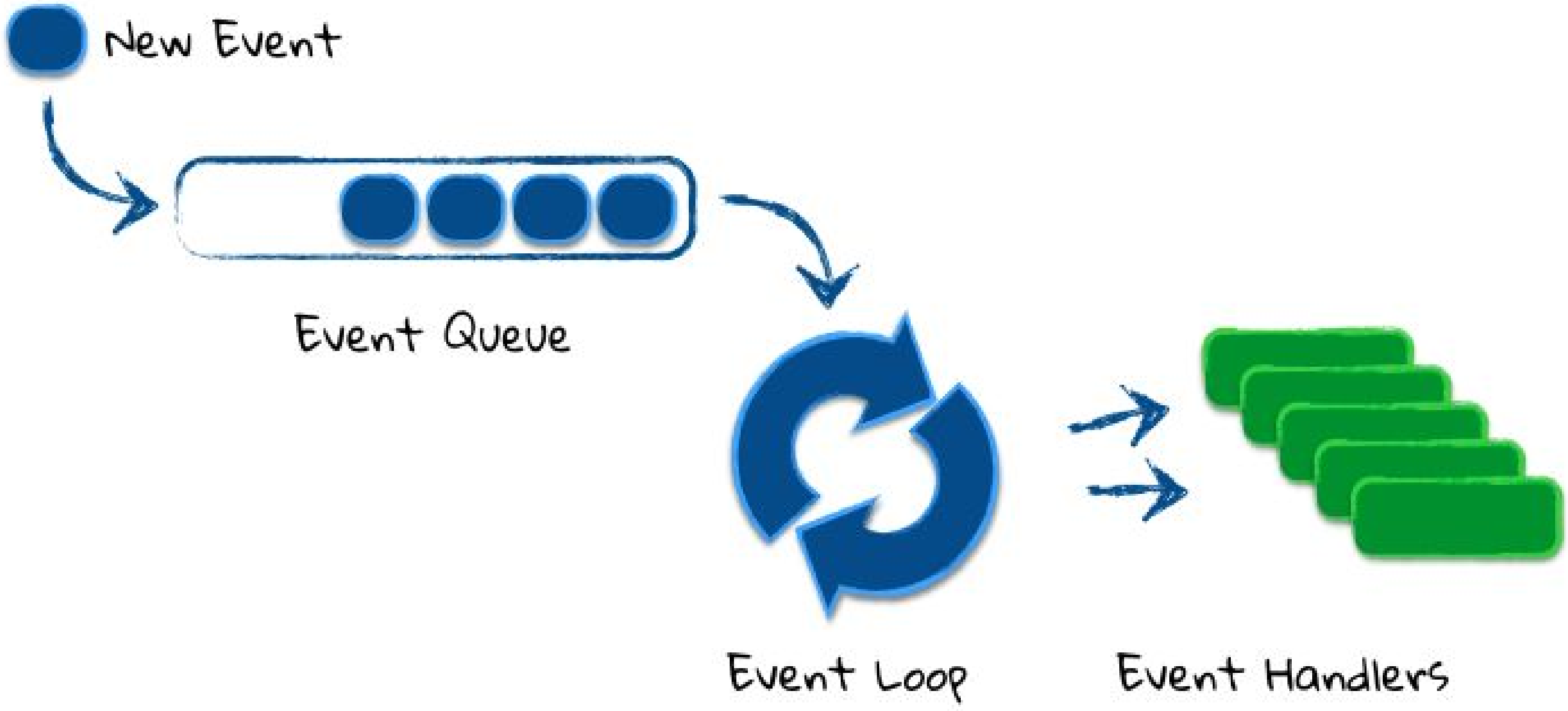
Communication Between Threads

- Threads are running asynchronously
- Can communicate through global variables and parameters
- Queue is often used for communication between threads

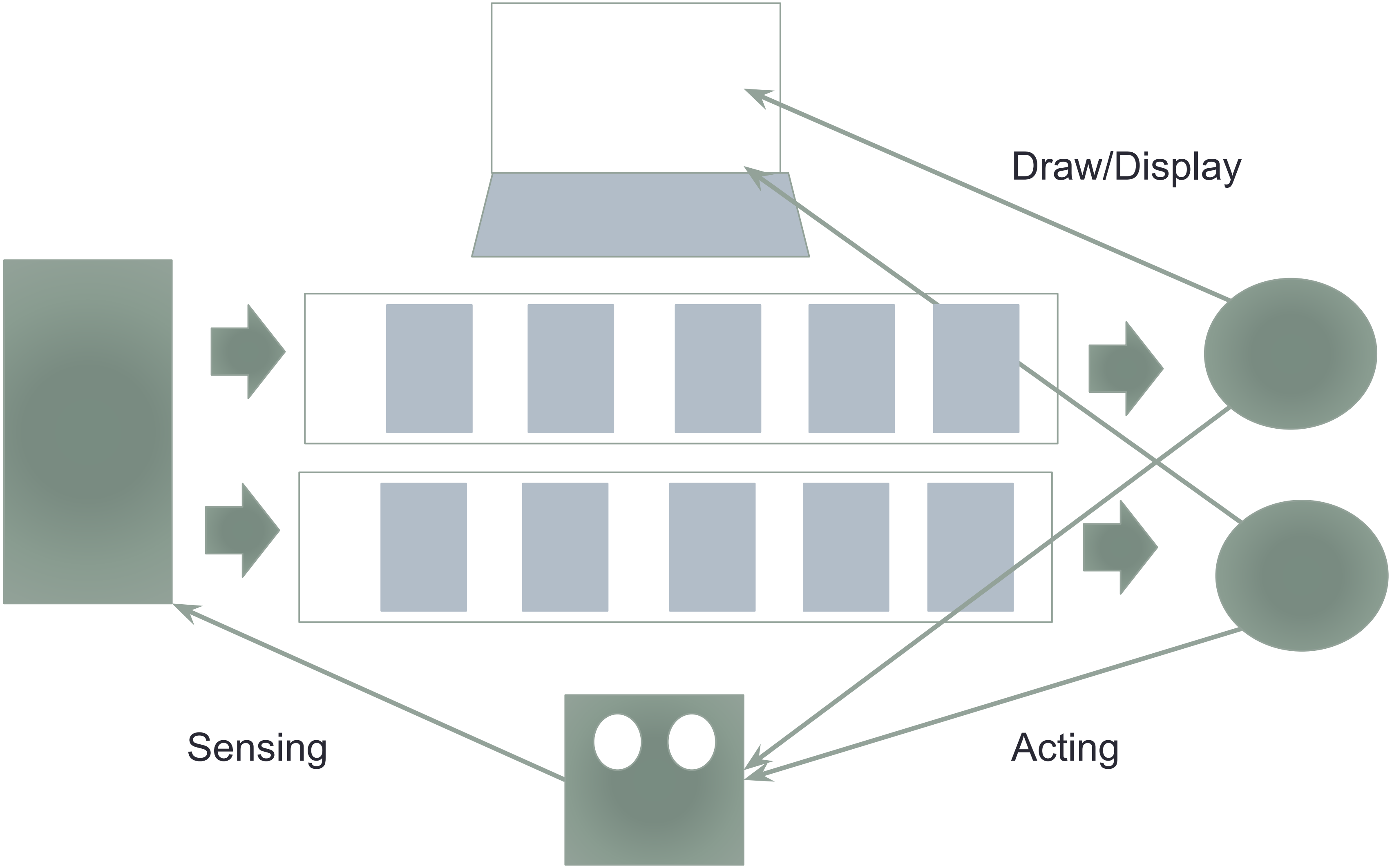
Different “types” of Queue

- FIFO queue:
  - *class* Queue.**Queue**(*maxsize=0*): *maxsize* is an integer that sets the upperbound limit on the number of items that can be placed in the queue.
- LIFO queue:
  - *class* Queue.**LifoQueue**(*maxsize=0*)¶
- Priority queue:
  - *class* Queue.**PriorityQueue**(*maxsize=0*)¶

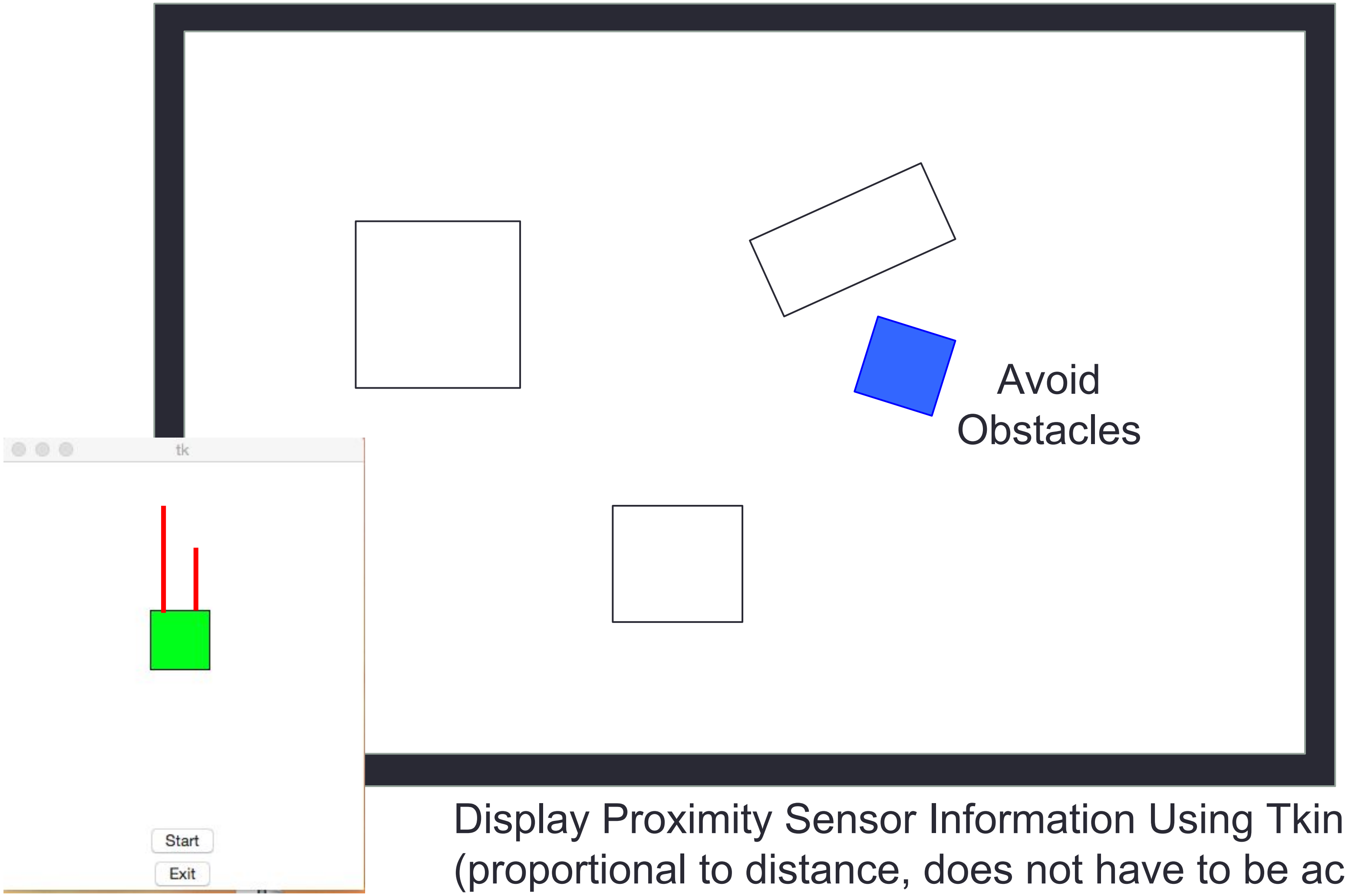
Event Queue



A Simple Structure Using Queues



Home Work #2-1: Escape



Avoid Obstacles

Display Proximity Sensor Information Using Tkinter (proportional to distance, does not have to be accurate)

Lec#07: Finite State Machine

- Concept: Finite State Machine (FSM)
  - What are FSM’s
  - Why / When to use FSM
- Implementation of Finite State Machines
  - FSM driven by an event queue
- Assignment#2-1: Escape

What Is A Finite State Machine

- A reactive system whose response to a particular stimulus (a *signal*, or a piece of *input*) is not the same on every occasion, depending on its current “state”.

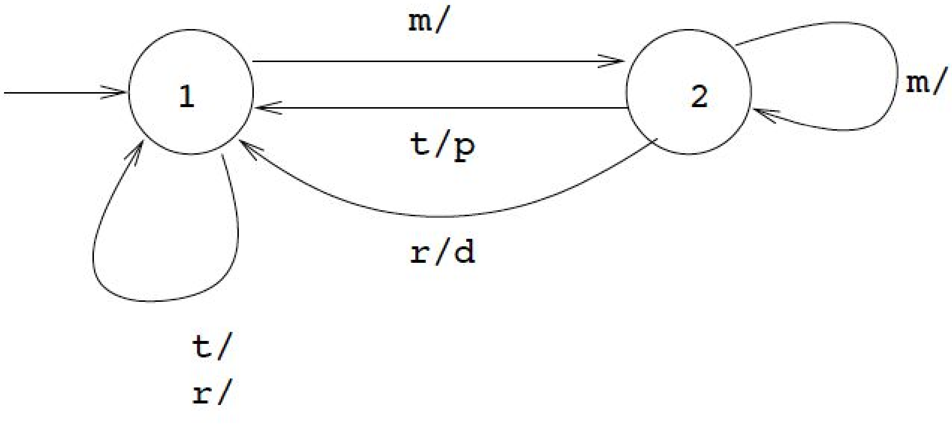
- For example, in the case of a parking ticket machine, it will not print a ticket when you press the button unless you have already inserted some money. Thus the response to the print button depends on the previous *history* of the use of the system.

More Precisely (Formally)

- A Finite State Machine is defined by $(\Sigma, S, s_0, \delta, F)$, where:
  - Σ is the input alphabet (a finite, non-empty set of symbols).
  - S is a finite, non-empty set of states.
  - s_0 is an initial state, an element of S.
  - δ is the state-transition function: $\delta : S \times \Sigma \rightarrow S$
  - F is the set of final states, a (possibly empty) subset of S.
  - O is the set (possibly empty) of outputs

A (Simplified) Ticket Machine

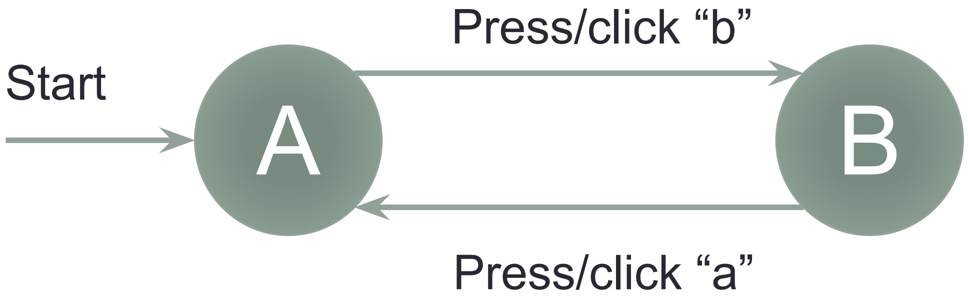
- Σ (m, t, r) : inserting money, requesting ticket, requesting refund
- S (1, 2) : unpaid, paid
- s_0 (1) : an initial state, an element of S.
- δ (shown below) : transition function: $\delta : S \times \Sigma \rightarrow S$
- F : empty
- O (p/d) : print ticket, deliver refund



How To Implement an FSM

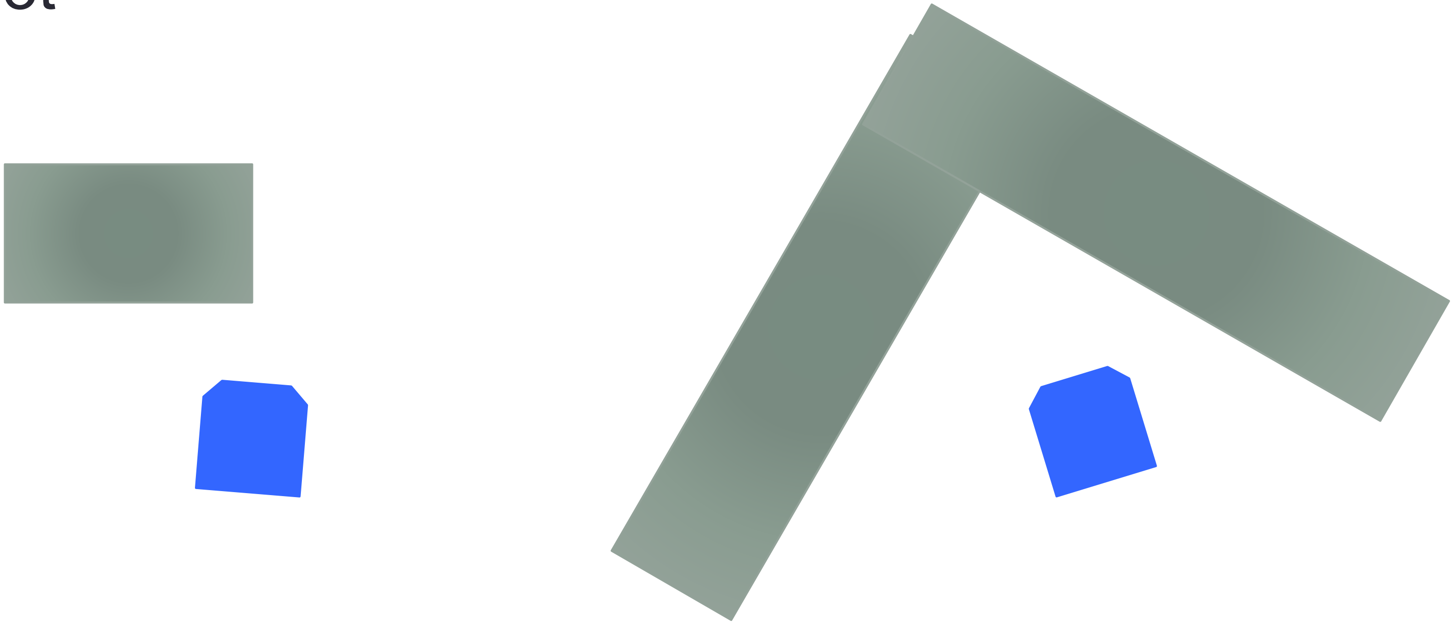
- The Finite State Machine class keeps track of the current state, and the list of valid state transitions.
- You define each transition by specifying :
  - FromState - the starting state for this transition
  - ToState - the end state for this transition
  - condition - a callable which when it returns True means this transition is valid
  - callback - an optional callable function which is invoked when this transition is executed.

Simplest FSM



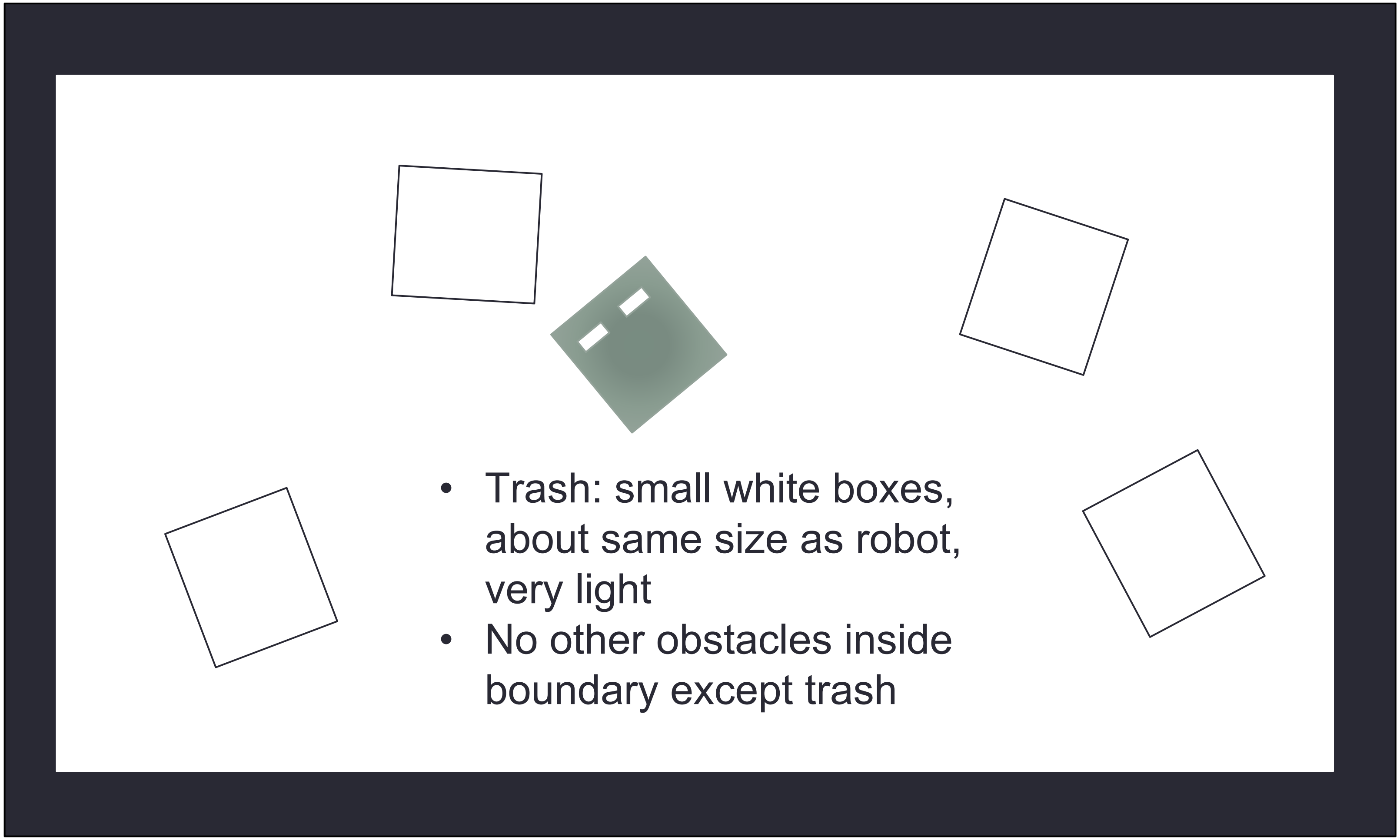
Why Finite State Machines For Robot

- Response to an event is dependent on the “state” of the robot



Turn-left, turn-right

Home Work #2-2: “Cleaner” (Push Out “Trash”)



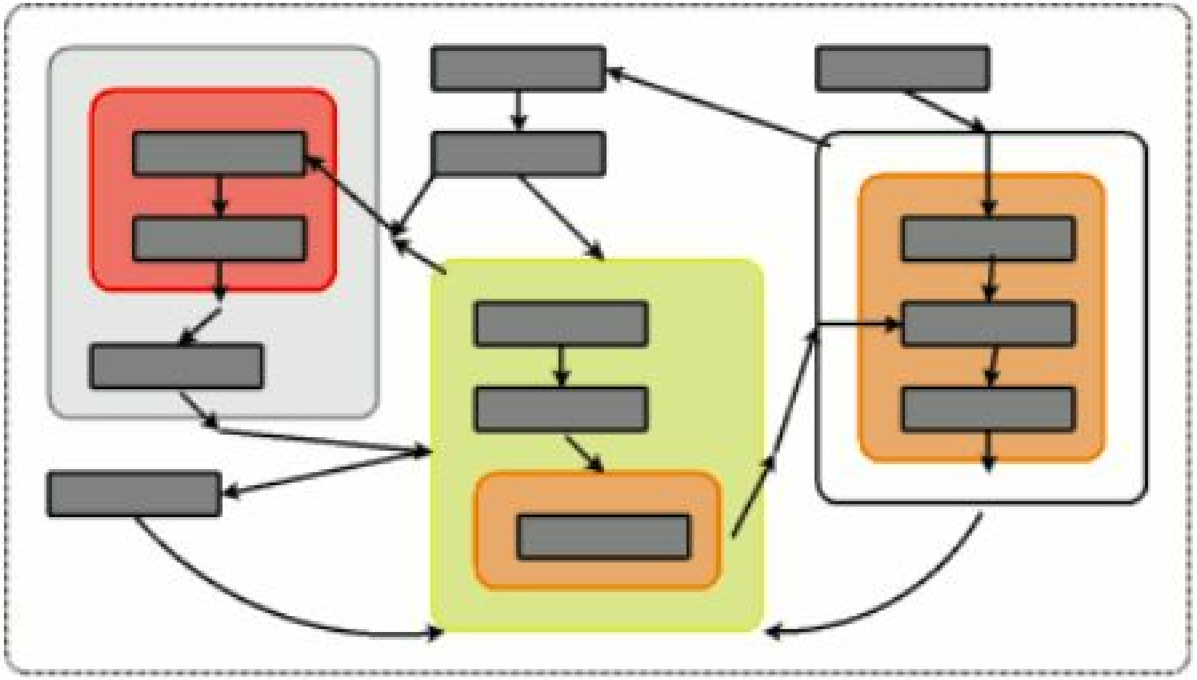
- Trash: small white boxes, about same size as robot, very light
- No other obstacles inside boundary except trash

Lec#08: HFSM & BT

- HFSM: Hierarchical Finite State Machine
- BT: Behavior Tree

Hierarchical Finite State Machine

- a.k.a StateCharts (first introduced by David Harel)



Harel’s StateCharts

- **Super-states** : groups of states.
  - These super-states too can have transitions, which allows you to prevent redundant transitions by applying them only once to super-states rather than each state individually.
- **Generalized transitions** : transitions between Super-states

Simplest Example

- Clustering / Super State

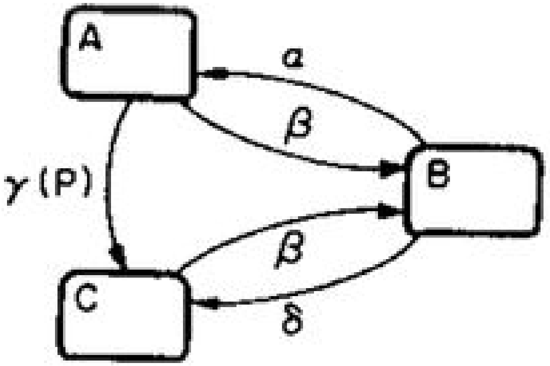


Fig. 1.

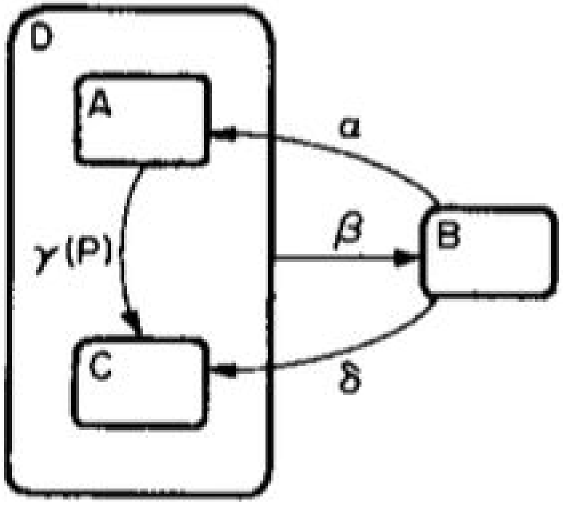
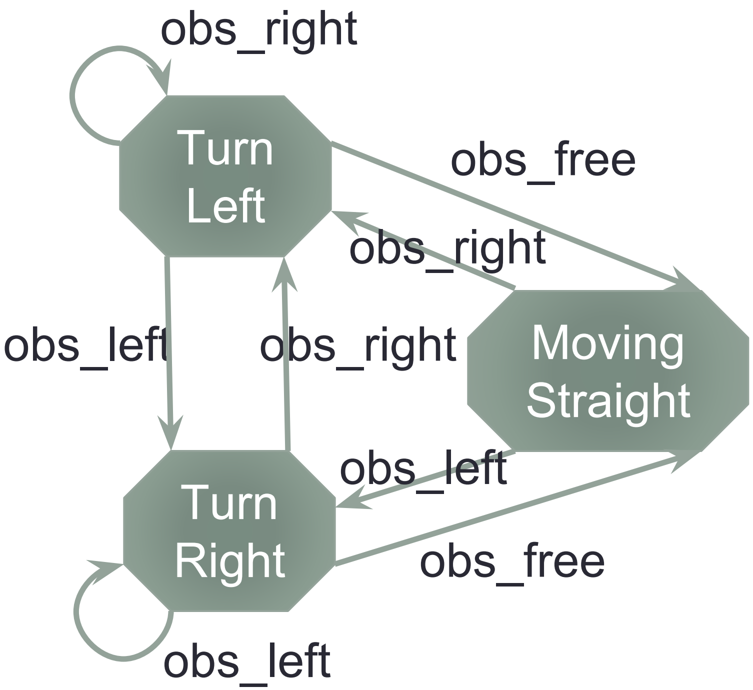
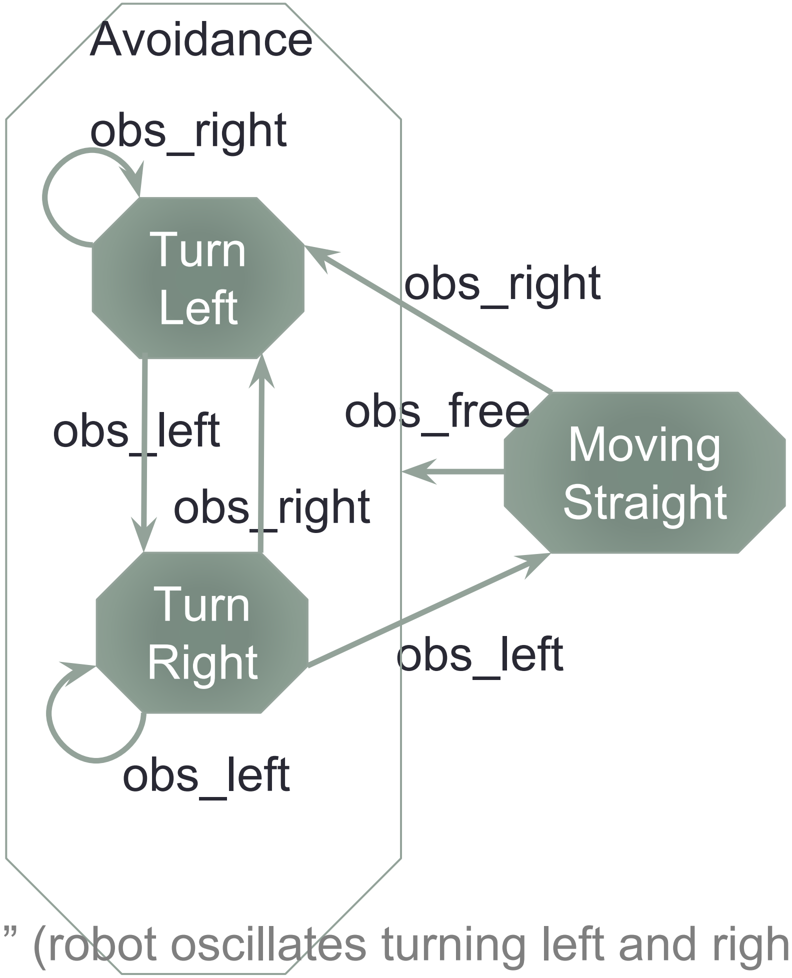


Fig. 2.

Obstacle Avoidance Example





Note: this algorithm can cause "oscillation" (robot oscillates turning left and right) in case of concave obstacle. But we discussed in class how to solve that

HFSM

- Refinement

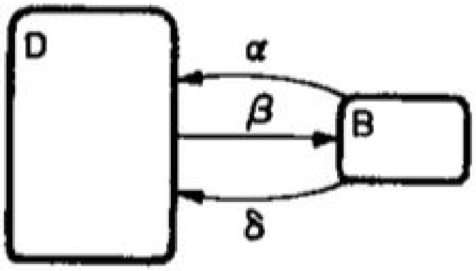


Fig. 3.

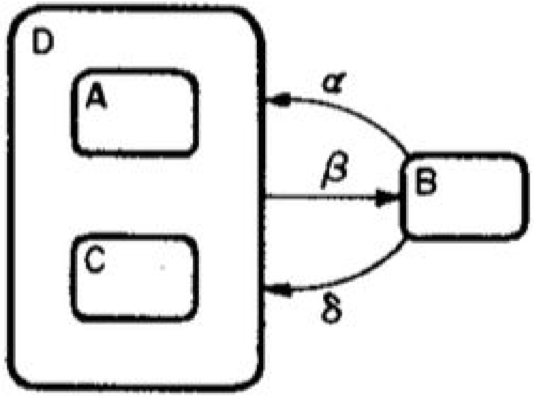


Fig. 4.

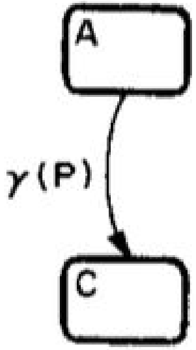


Fig. 5.

Behavior Trees (BT)

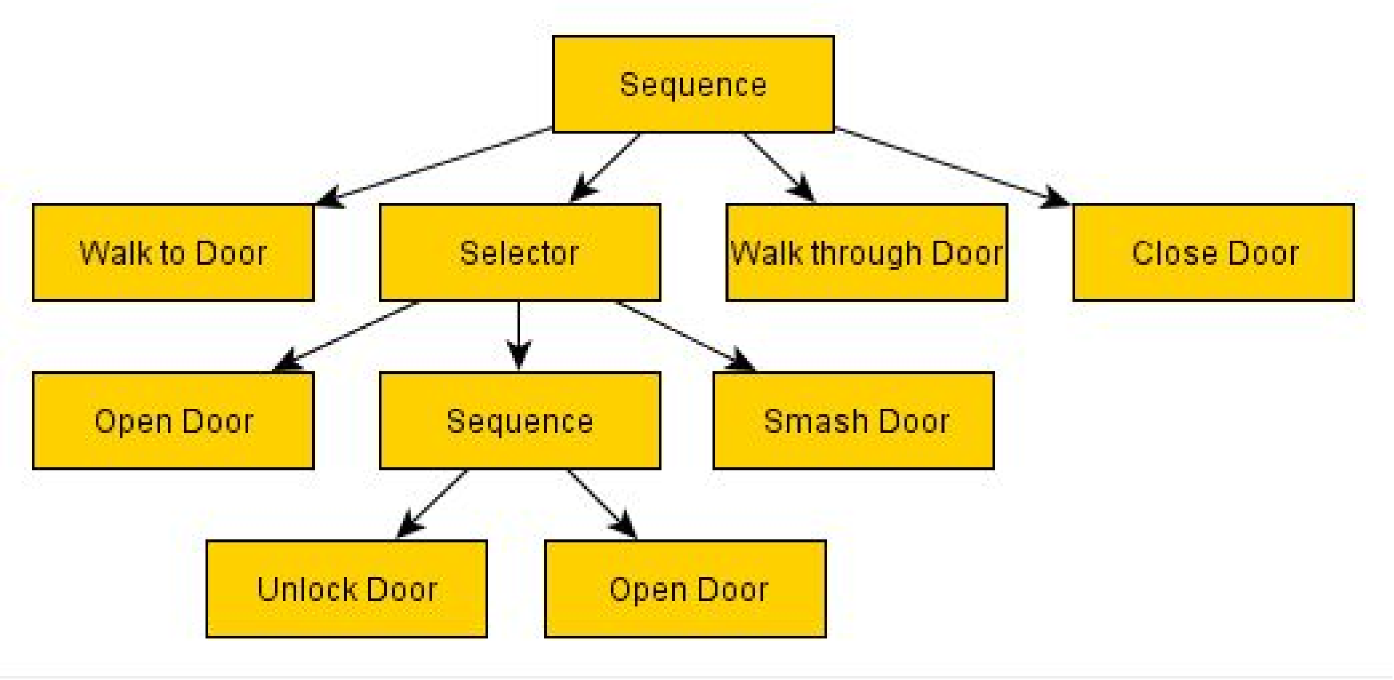
- Mathematical Model of Plan Execution – describe switching between a finite set of tasks in a modular fashion
- Originated from Game Industry, as a powerful way to describe AI for “NPC”
  - Halo, Bioshock, Spore

More Formally (Precisely)

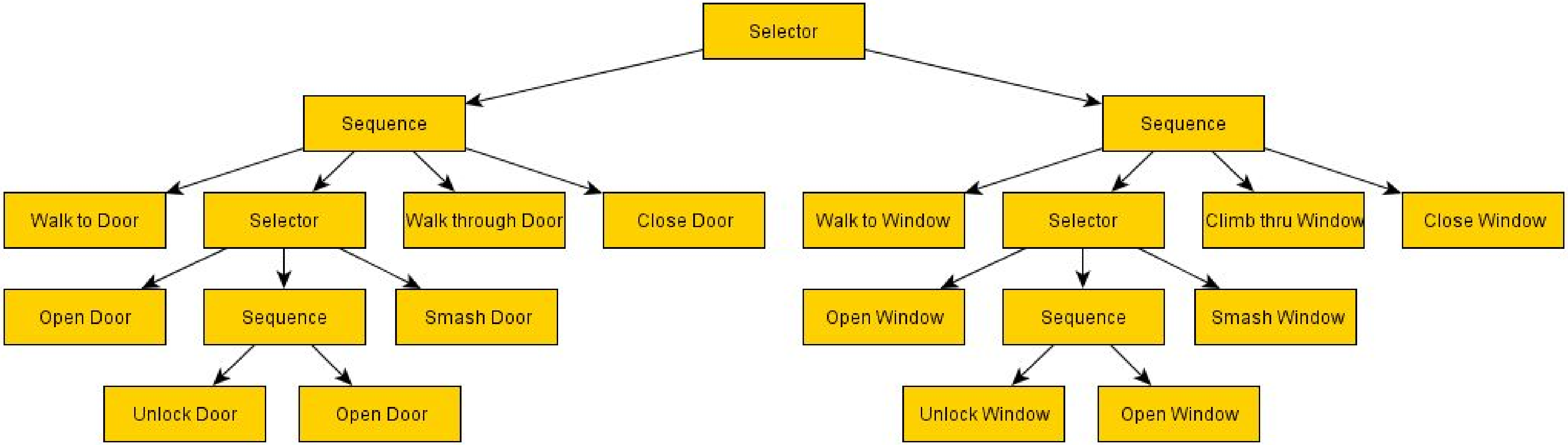
- Directed Acyclic Graph
- Four types of nodes:
  - **Root node** – no parent, one child (ticks)
  - **Composite node** (“Control flow ”) – one parent, and one or more children
  - **Leaf node** (“Execution”) – one parent, no child (Leaves)
  - **Decorator node** (“Operator”) – one parent, one child

BT Execution

- Depth-First Traversal



BT Execution



Topics For Part 3

3.1 The Robot Programming Problem

- What is “robot programming”
- Challenges
- Real World vs. “Virtual” World
    - Mapping and visualizing Hamster’s world
- A decomposition of the “mobile robot programming” problem

3.2 “Modeling” Hamster

- Hamster’s Motion and Sensors

3.3 Localization

- Where am I?
- Sub-goal navigation

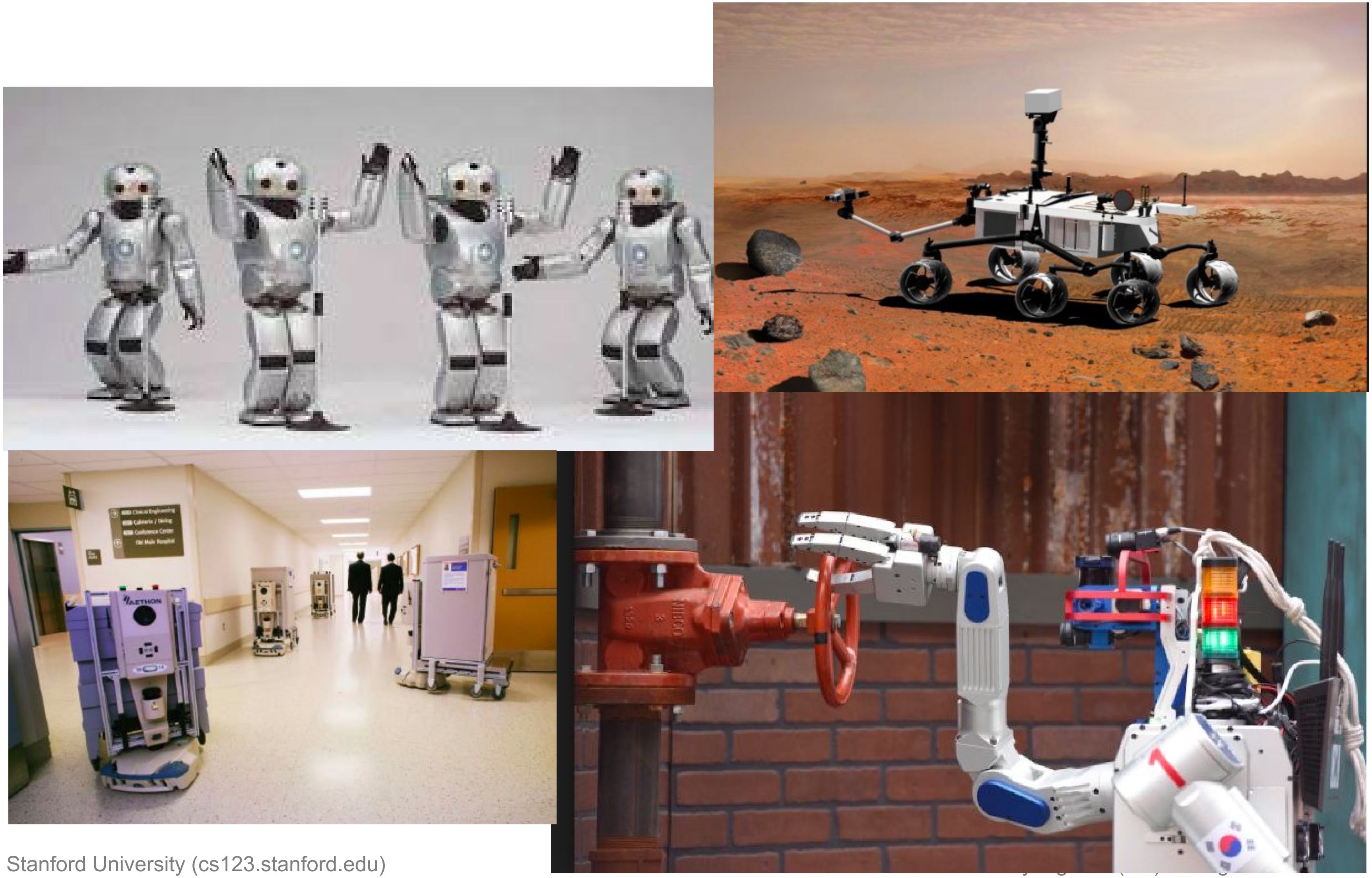
3.4 Plan and Execution

- Motion Planning & Control with Uncertainty

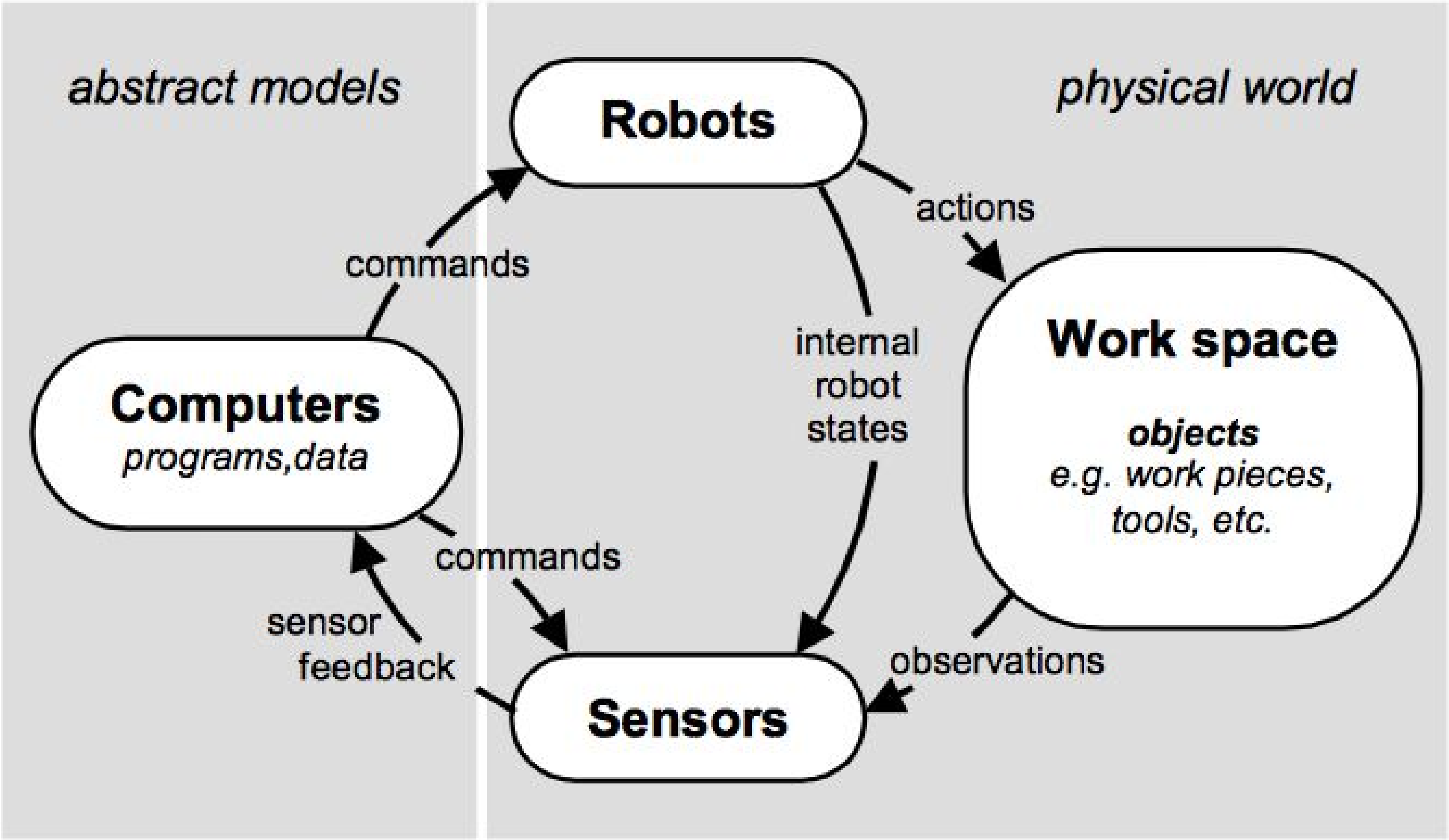
Lec#09: Reasoning w/ Uncertainty

- Part 3-1: Challenges of Robot Programming
- What is robot programming
  - Modeling
  - Localization
  - Planning
  - Execution
  - Reactive is not enough: better knowledge of environment
- Physical world vs. virtual world
  - Modeling of Hamster: physical vs. virtual world
  - What does the robot see
  - How to make sense of what the robot see
- Graphic toolkit to help you visualize Hamster
- Assignment#3-1: Localization

What Is Robot Programming



A Simplified Paradigm



Virtual World

Real (Physical) World

Basic Elements Of Robot Programming

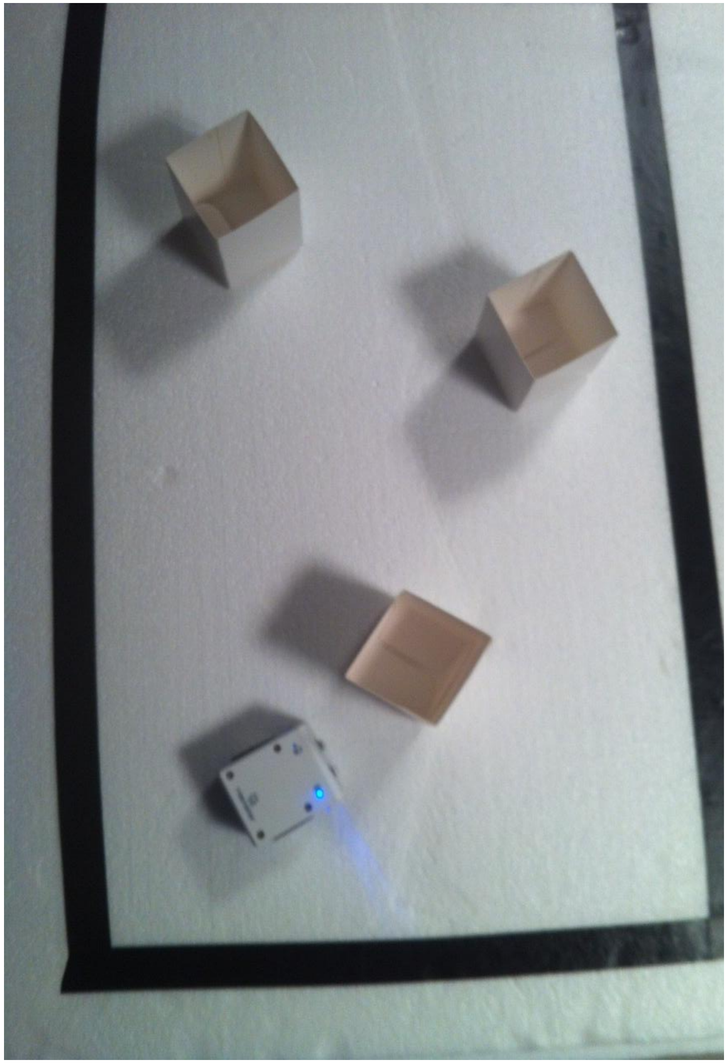
- Model of itself
- Model of the world (mapping virtual world and real world)
- Description of a task
- Description of a "plan" (to achieve task)
  - can be given to the robot
  - can be generated by robot
- A way to recognize success (task completion)
  - and monitoring during plan execution to make sure it's following the plan

Unique Challenges

- Knowledge of the world incomplete
  - Not available
  - Impractical (too much details)
  - World Changing
- Sensing is imperfect
  - And limited
- Control is inaccurate

Trash Cleaning Example

- Model of itself
- Model of the world
- Description of a task
- Description of a “plan” (to achieve task)
  - can be given to the robot
  - can be generated by robot
- A way to recognize success (task completion)
  - monitoring during plan execution to make sure it’s following the plan



Reactive Is Not Enough

So far we have:

- Very limited knowledge of the world (border and obstacles exist)
- Only “reactive” behaviors

But you can not do too much being completely “reactive”

To do more:

- we need better “knowledge” of the world and
- use this knowledge to generate a “plan”
- ensure “plan” execution

Lec#10: Localization

- Localization
  - Relative (Internal): dead reckoning
  - Absolute (External): distance sensors (Geometric feature detection), IR, Landmark
- Modeling Environment
  - Least Square (Fit): minimization
- Assignment#3-1 – Localization

Localization Methods

Two General Approaches:

- Relative (Internal) – relative to “self”
  - Using Proprioceptive sensors such as:
  - odometric (encoder)
  - gyroscopic
- Absolute (External)
  - using “exteroceptive” sensors such as infrared, sonar, laser distance sensor – to measure environment
  - geometric features
  - landmarks

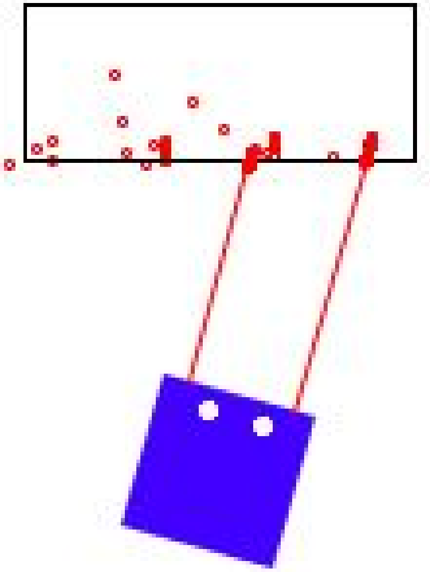
Relative “Localization”: Dead Reckoning

- What is Dead Reckoning
- Encoder
- Various Drive Mechanisms
- Hamster

“Absolute” Localization

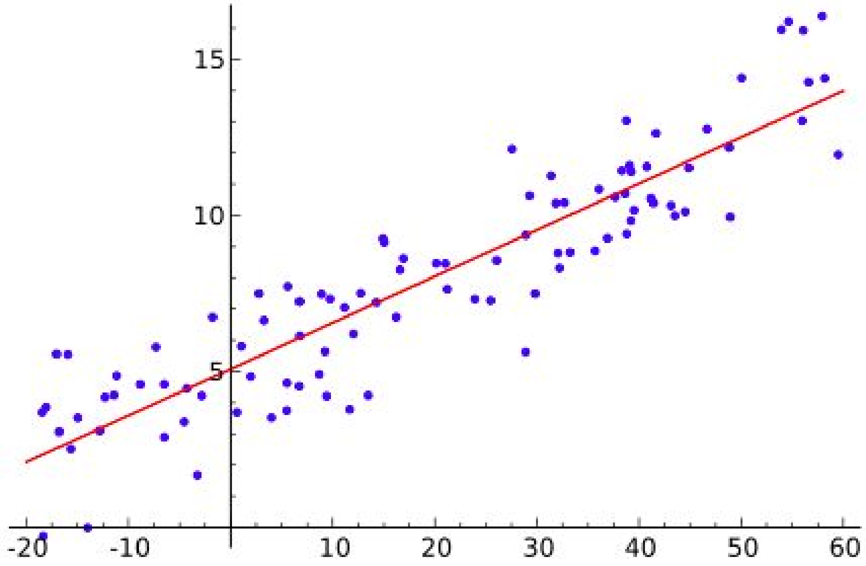
- GPS and Beacons
- Use “external” sensors – “measuring” environment and matching against “map”
- Minimize the difference between measured data and “expected” (predicted) data (from the map)

Making Sense of Noisy Data

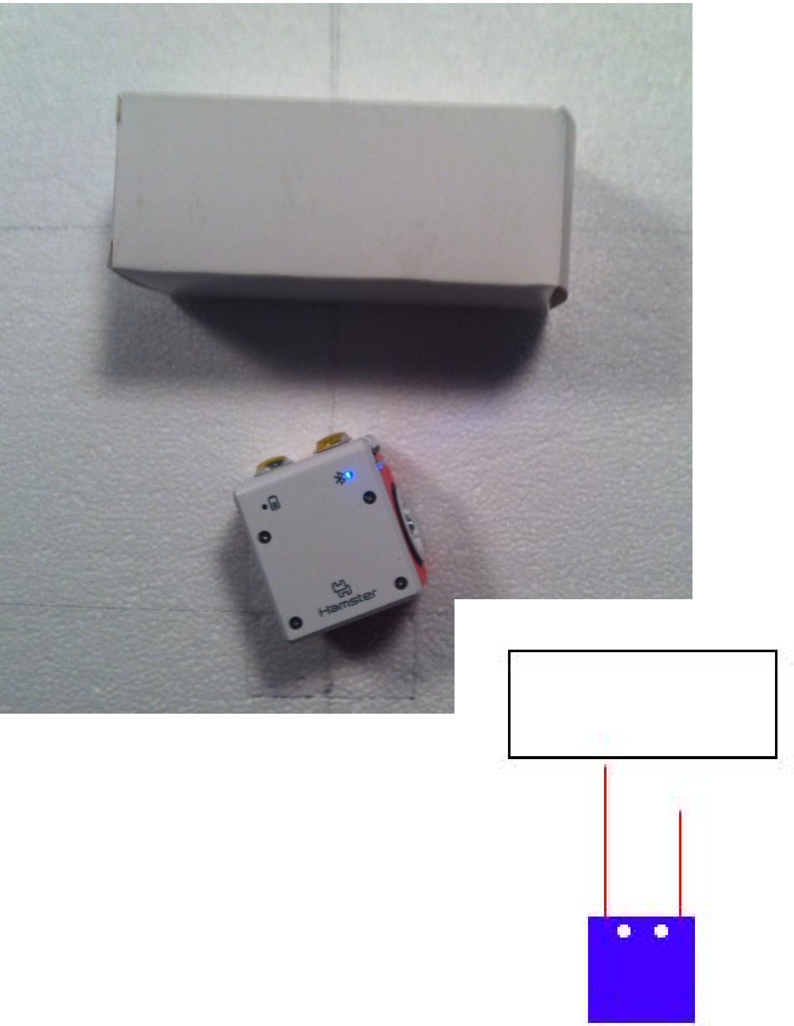


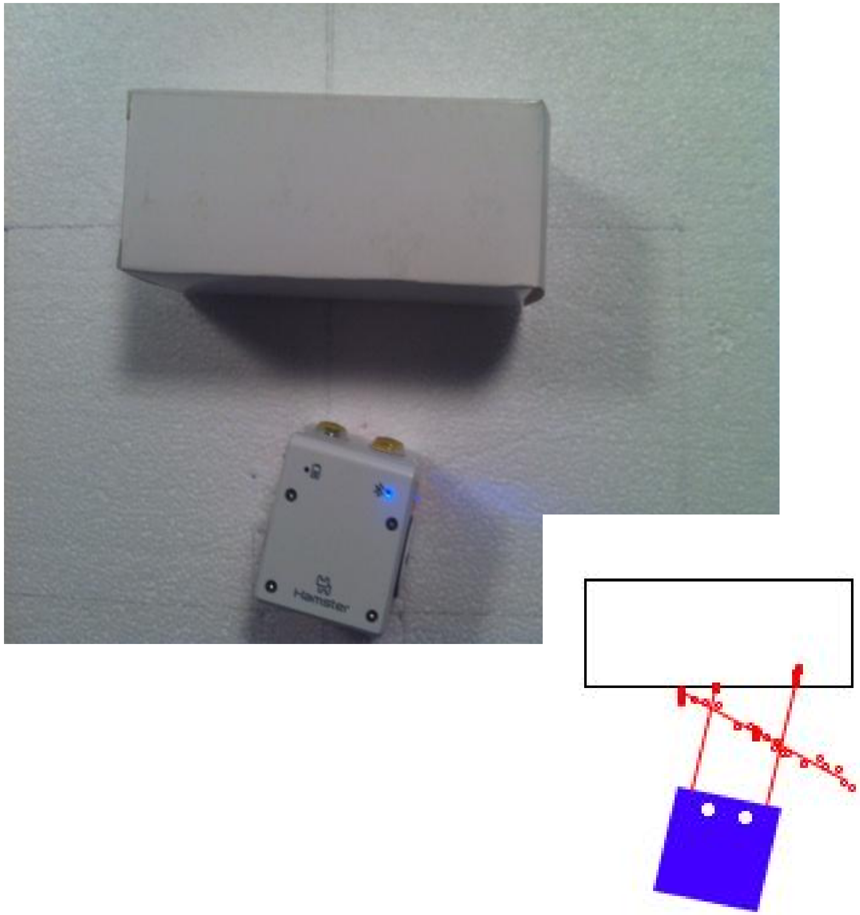
Linear Least Square (Fit)

- For a given set of points (x_i, y_i)
- Find m,c such that the sum of distances of these points to the line y = mx +c is minimized

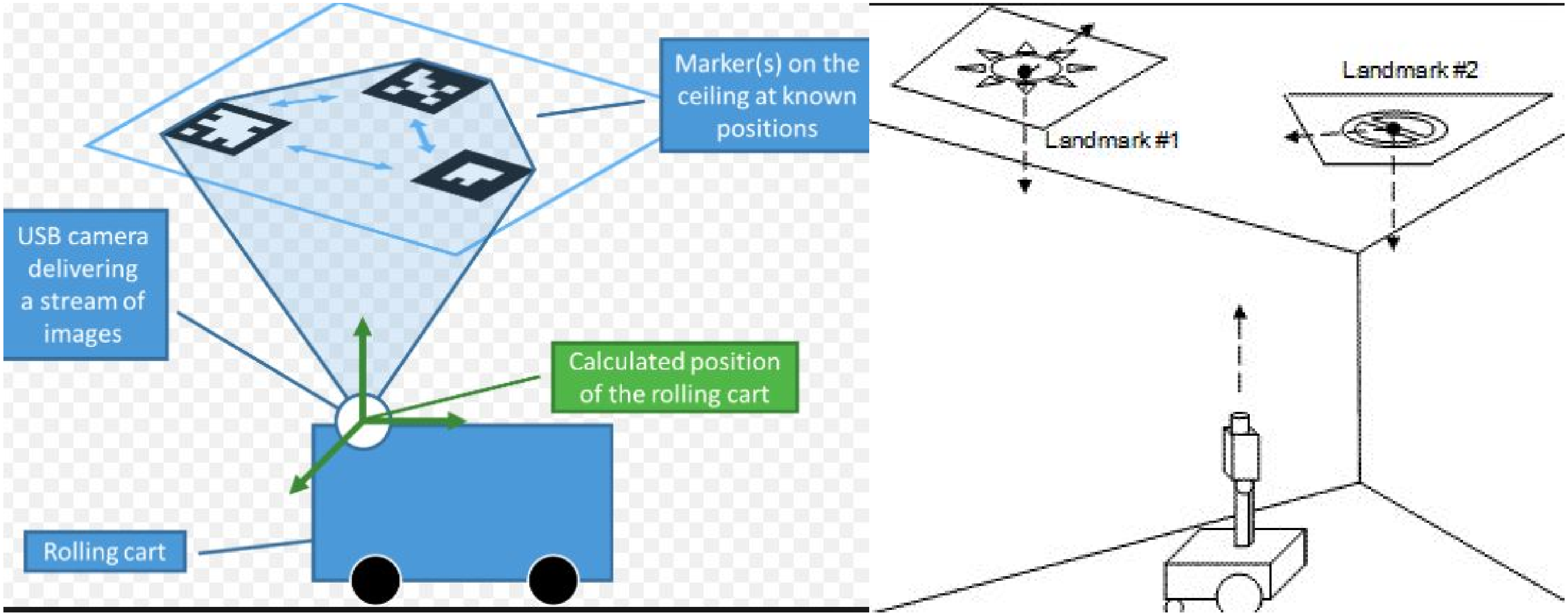


Localization Of Hamster



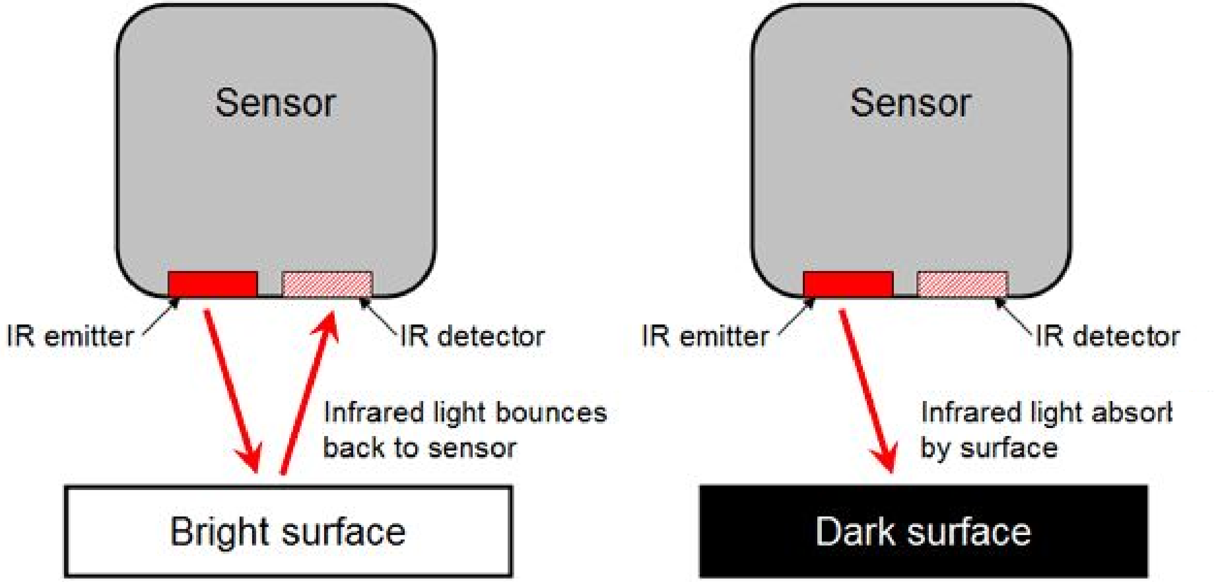


Localization Using Special Landmarks

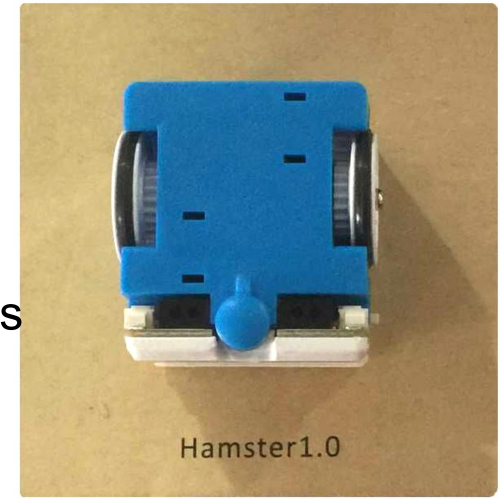


Patterns on ceiling are often used landmarks

Hamster “Floor” Sensors

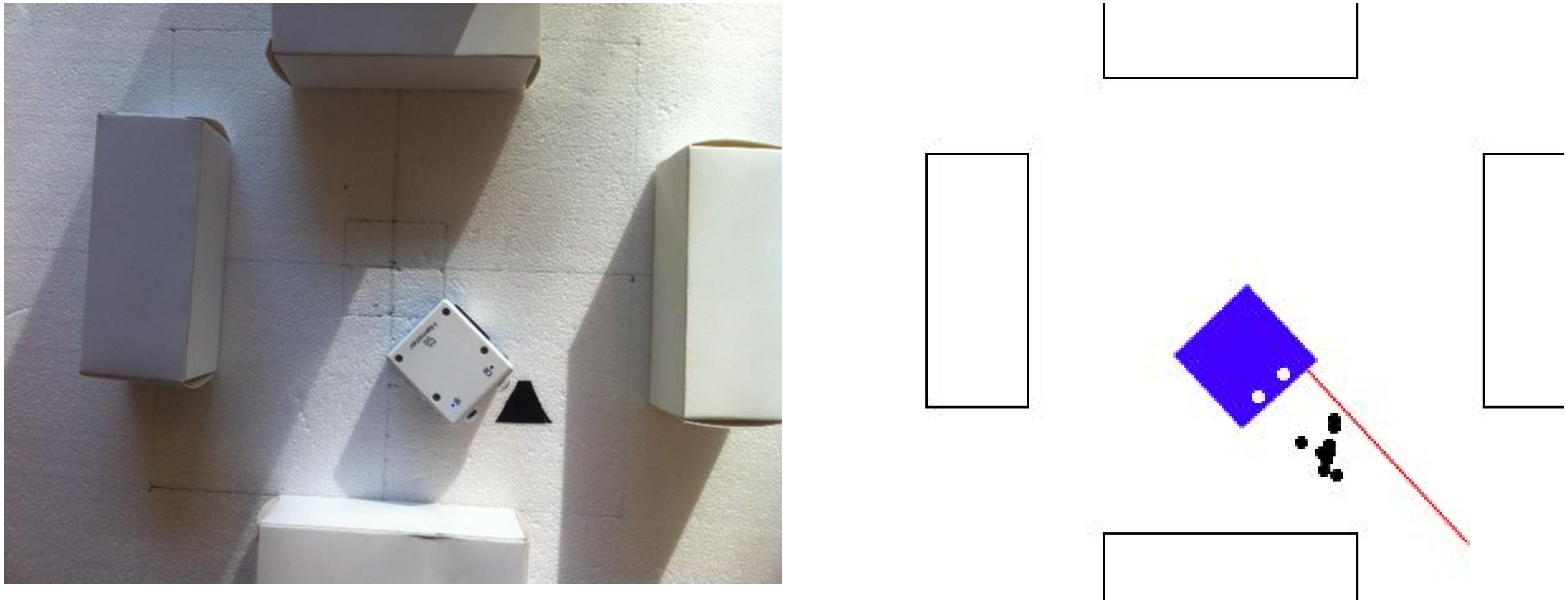


Left and Right Floor Sensors



Landmark Navigation Using Floor Sensors

- Greyscale
- Patterns



Combining Relative and Absolute Localization

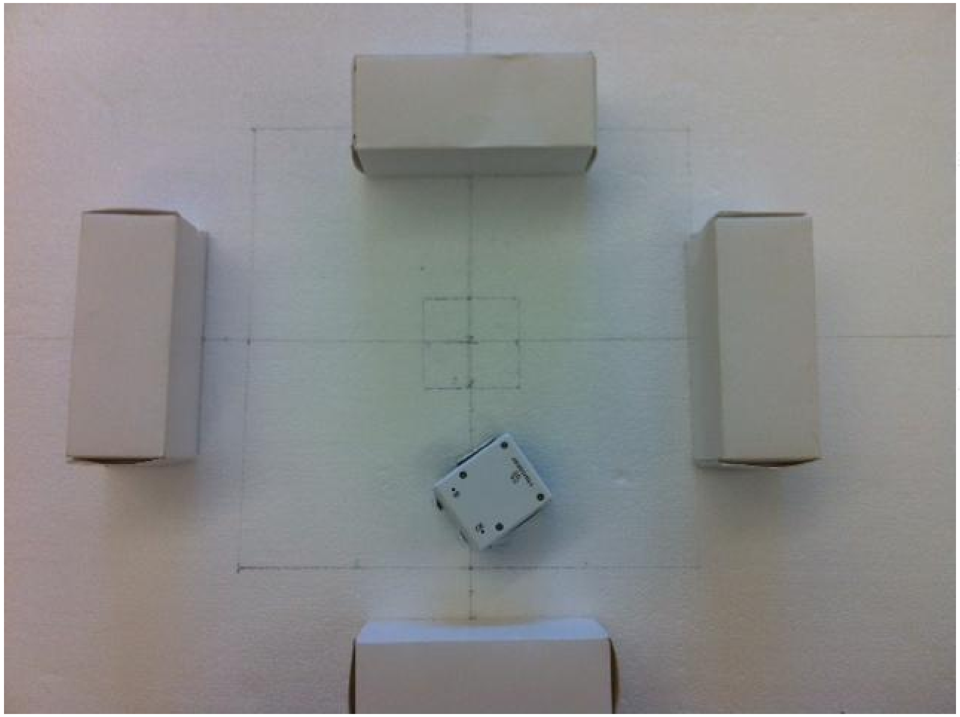
Dead reckoning +
Geometric feature based localization

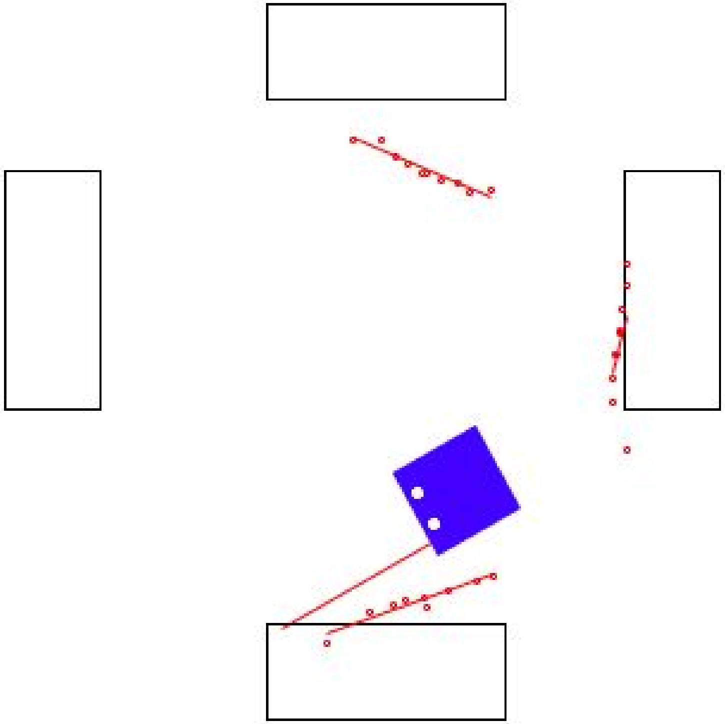
Mobile Robot Programming: Problem Decomposition

- Physical -> Virtual World Mapping
- Localization (Hamster knowing “where he is”)
- Local navigation (going to a specific place / location) : achieving “sub-goal”
- Plan and Plan Execution (execution monitoring)

Homework Part #3-1

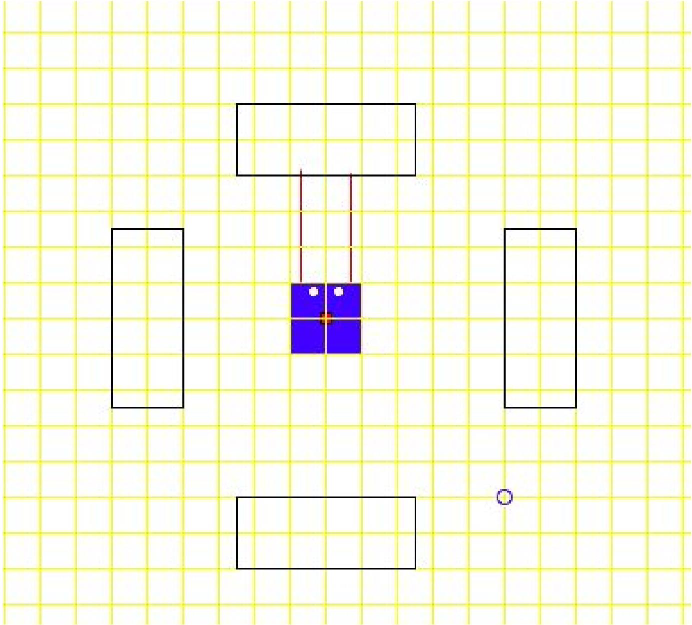
- Joystick your robot to face the obstacle on the different obstacles, and localize with respect to each

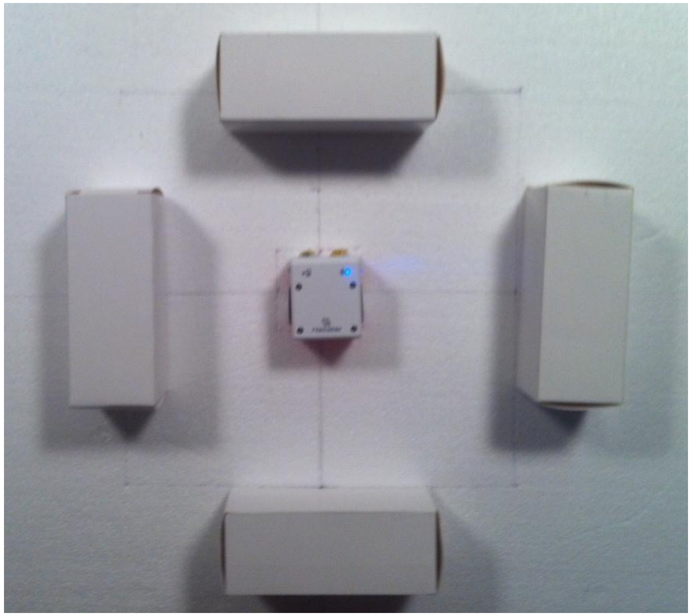




Homework #3-1:
"Local" Localization and Navigation

- Base on local (spatial and temporal) information
- Technique will be discussed on Thursday
- But you can first do the "robot modeling" part



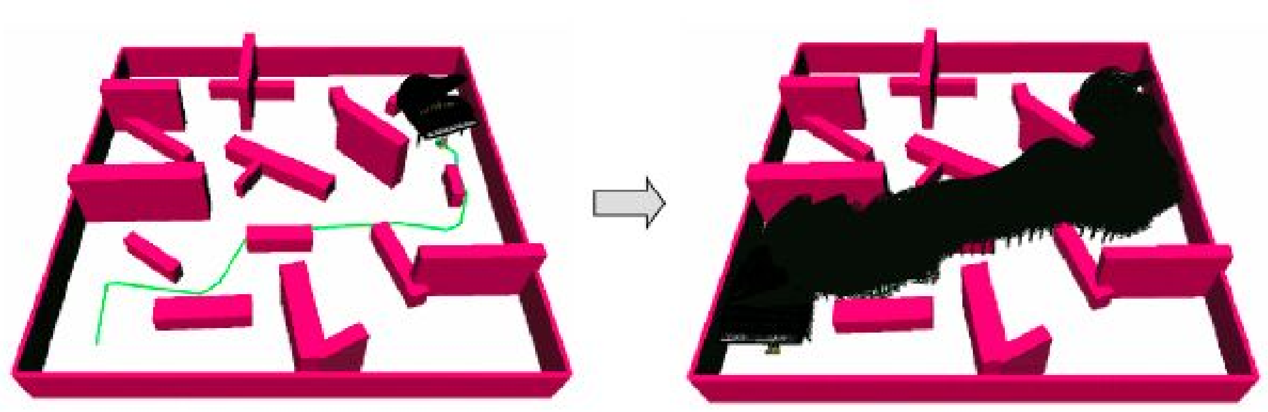


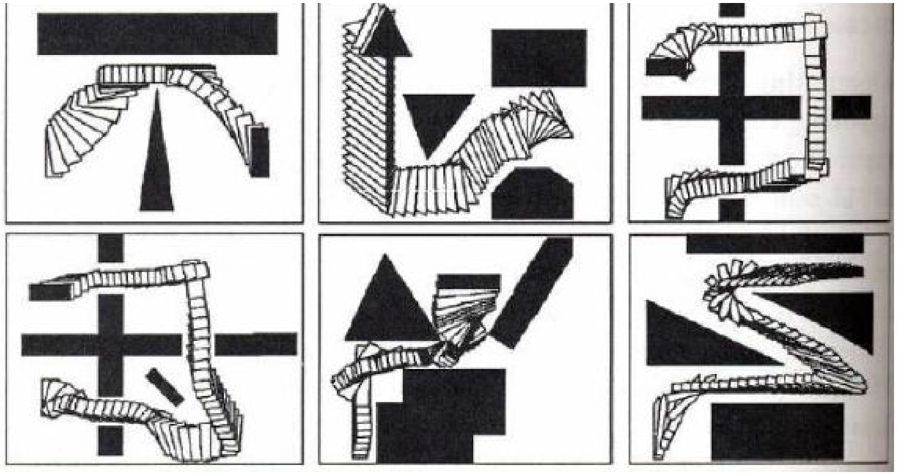
Lec#11: Motion Planning

- Introduction to Robot Motion Planning
  - Configuration Space (C-Space) Approach
  - Basic Motion Planning Methods: Discretization
    - Visibility Graph, Voronoi Diagrams
    - Cell Decomposition: Exact, estimate
- Plan Execution (Control)
  - Virtual World (Perfect Control)
  - Real World (Uncertainty in control)
- Planning Under Uncertainty
  - Landmarks
  - Preimage backchaining
- Homework Assignment Part #3-2

What is Motion Planning

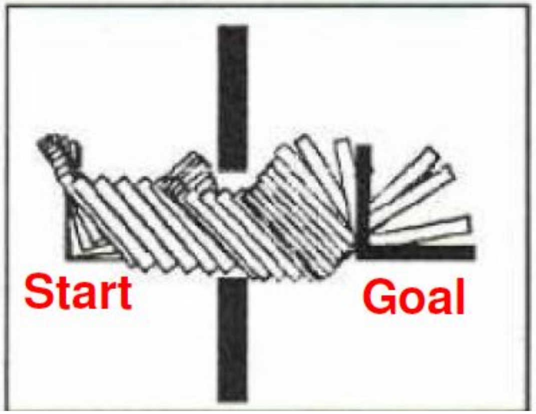
- Also known as the **Piano Mover**’s Problem

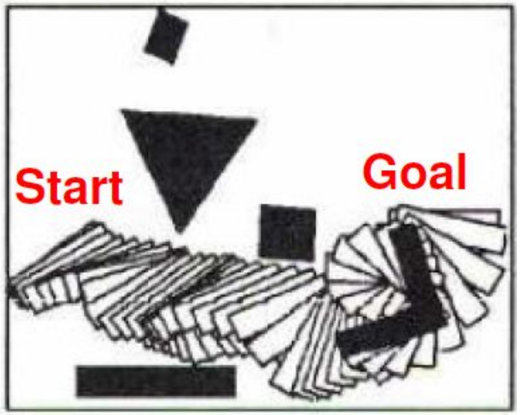




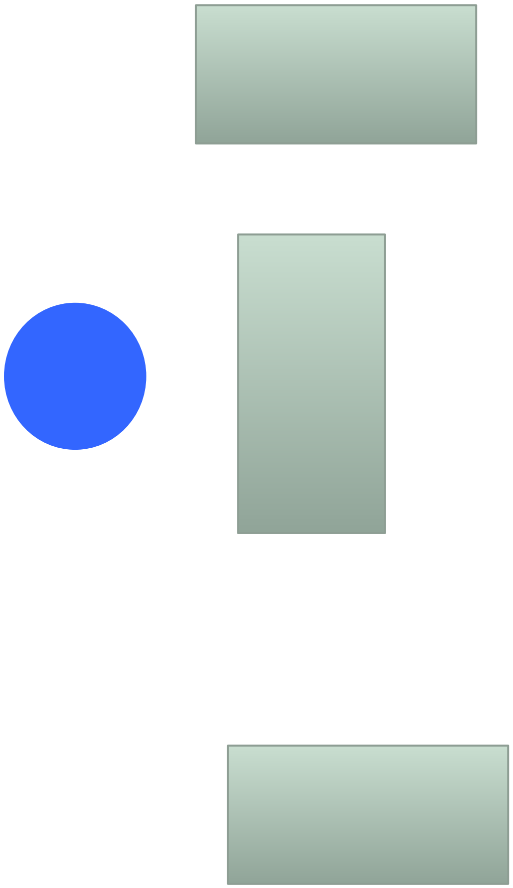
Problem Formulation

- The problem of motion planning can be stated as follows
  - A start pose of the robot
  - A desired goal pose
  - A geometric description of the robot
  - A geometric description of the world
- Find a path that moves the robot
  - from start to goal while
  - never touching any obstacle

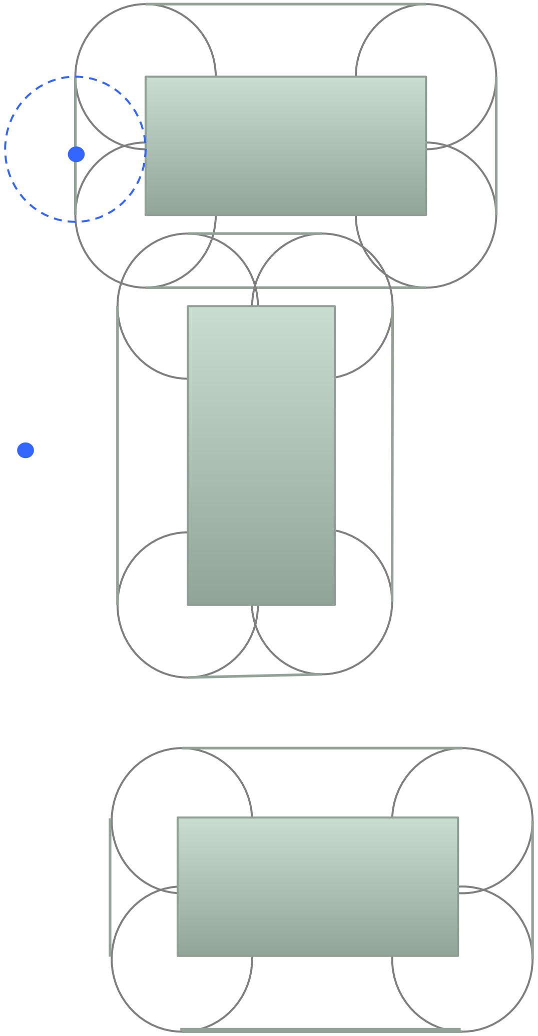




Example of 2D Circular Robot



Work Space



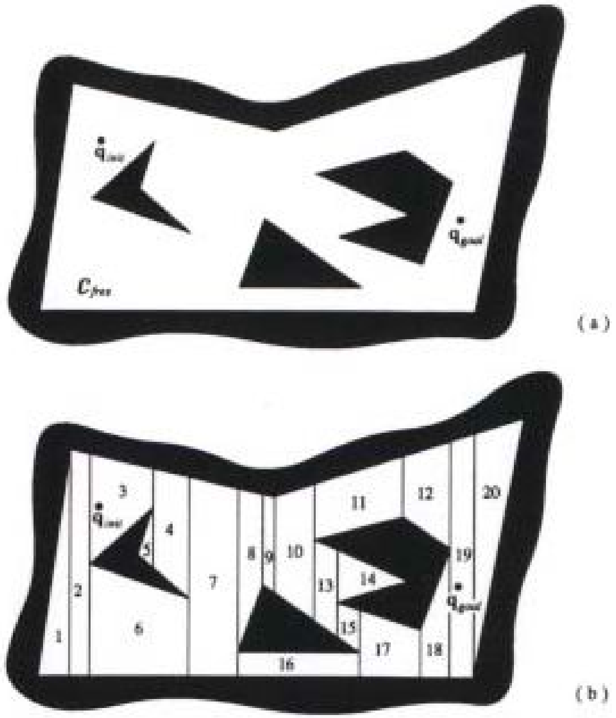
Configuration Space

Motion Planning Methods

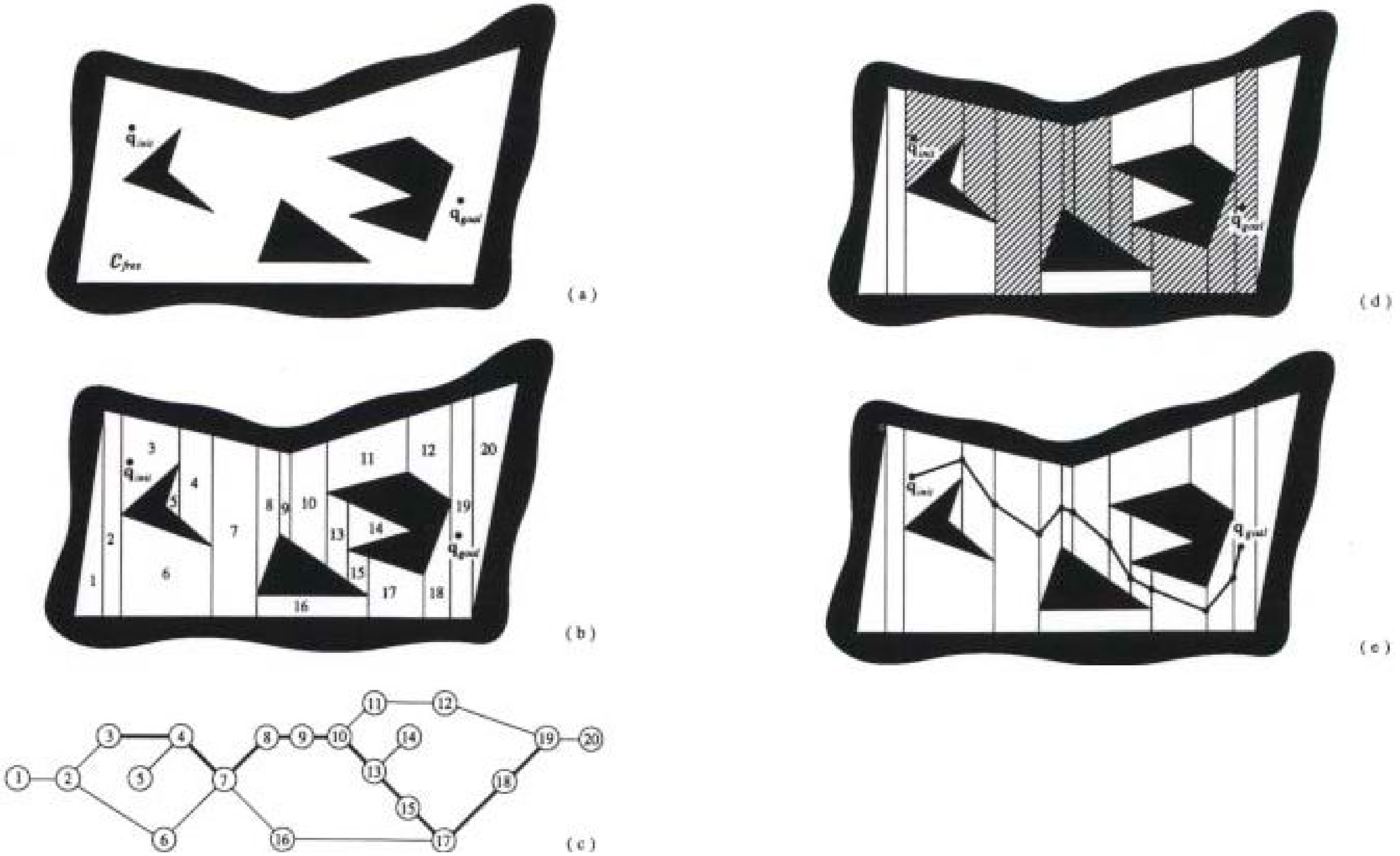
- Converting a “continuous” space problem into a discrete graph search problem (discretization of C-space)
- Decouple “independent” DoF
  - mobile vs. manipulation
- We will focus on planning problem of mobile robots
- Visibility Graph
- Voronoi Diagrams
- Cell Decomposition
  - Exact
  - Approximate

Motion Planning: Discretization of Space

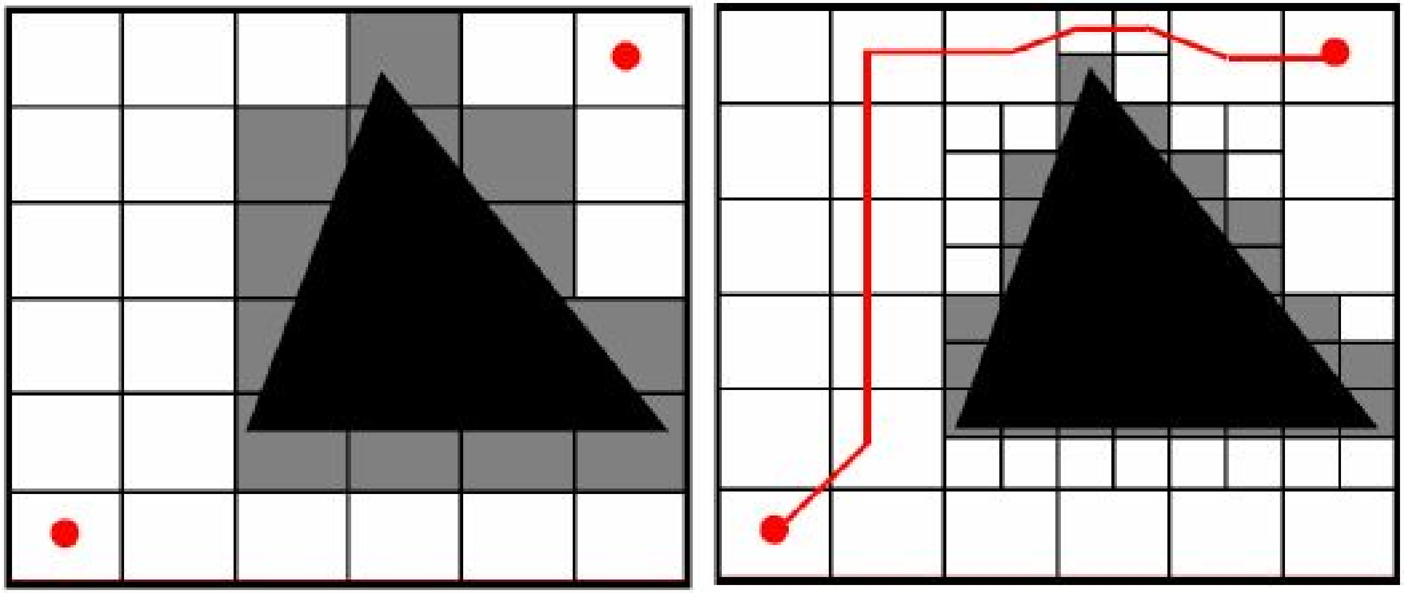
- Different methods for “discretizing” space:
  - Visibility Graph
  - Voronoi Diagram
  - Cell Decomposition



Cell Decomposition : Exact



Cell Decomposition : Approximate

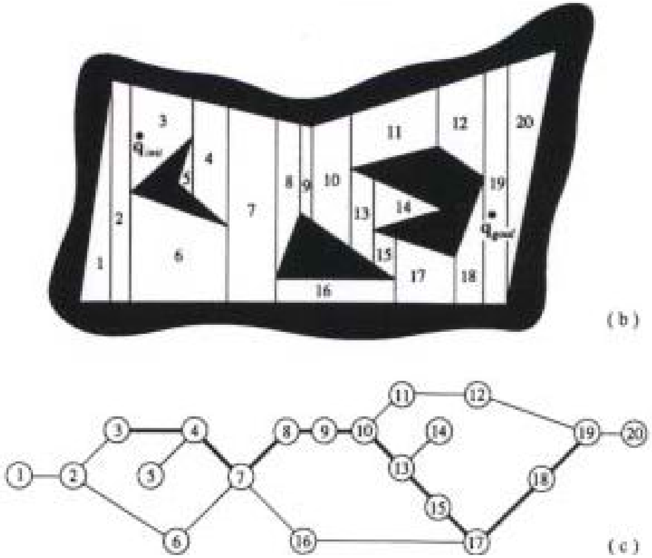


Search

- Uninformed Search
  - Use no information obtained from the environment
  - Blind Search
    - BFS (Breath First)
    - DFS (Depth First)
- Informed Search
  - Use evaluation function
  - Use “Heuristic” to guide the search:
    - Dijkstra’s Algorithm
    - A*

Use of Heuristics

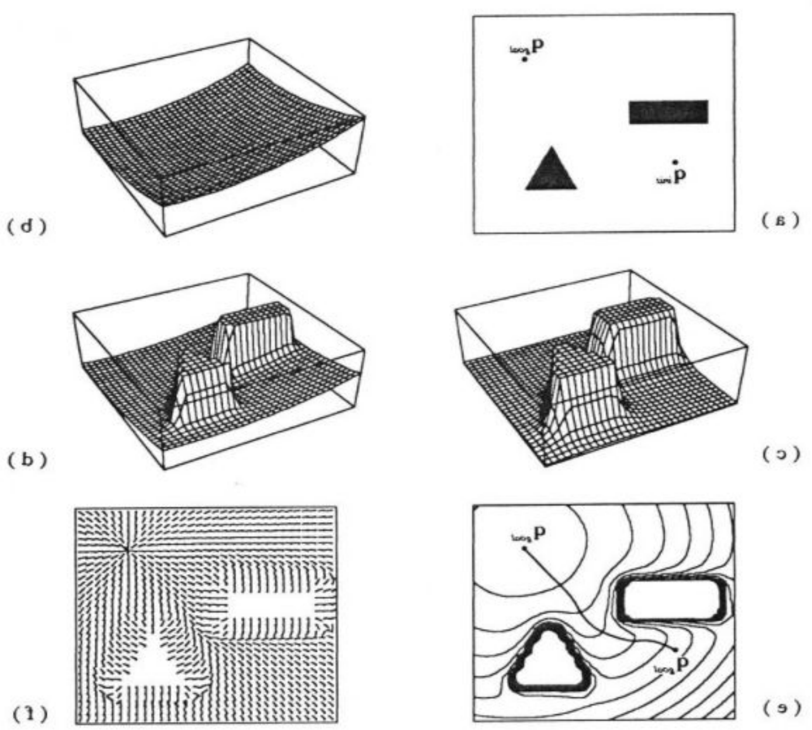
- Estimate "Distance to Goal" at each node



Potential Field Method

- All techniques discussed so far aim at capturing the connectivity of *C_free* into a graph
- **Potential Field Methods** follow a different idea:
  - The robot, represented as a point in *C*, is modeled as a **particle** under the influence of a **artificial potential field U** which superimposes
    - **Repulsive forces** from obstacles
    - **Attractive force** from goal

Potential Field Method: Gradient Descent

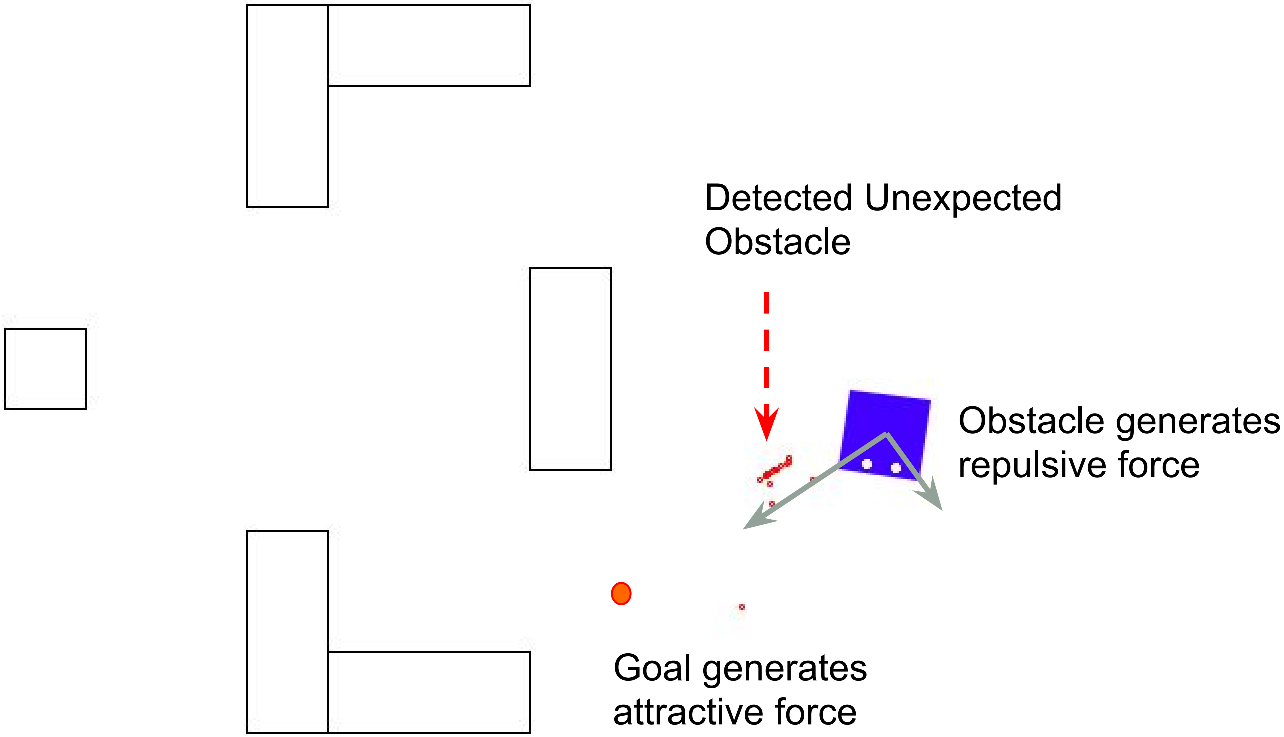


16-735, Howie Choset, with slides from Ji Yeong Lee, G.D. Hager and Z. Dodds

“Unexpected” Obstacle Avoidance

- Simple Potential Field Method has the drawback of getting stuck at “local minimum”
- But is good for “local obstacle” avoidance, such as
  - unexpected obstacles in environment (like moving people)
  - or known obstacle become “unexpected” due to control uncertain

Local Obstacle Avoidance



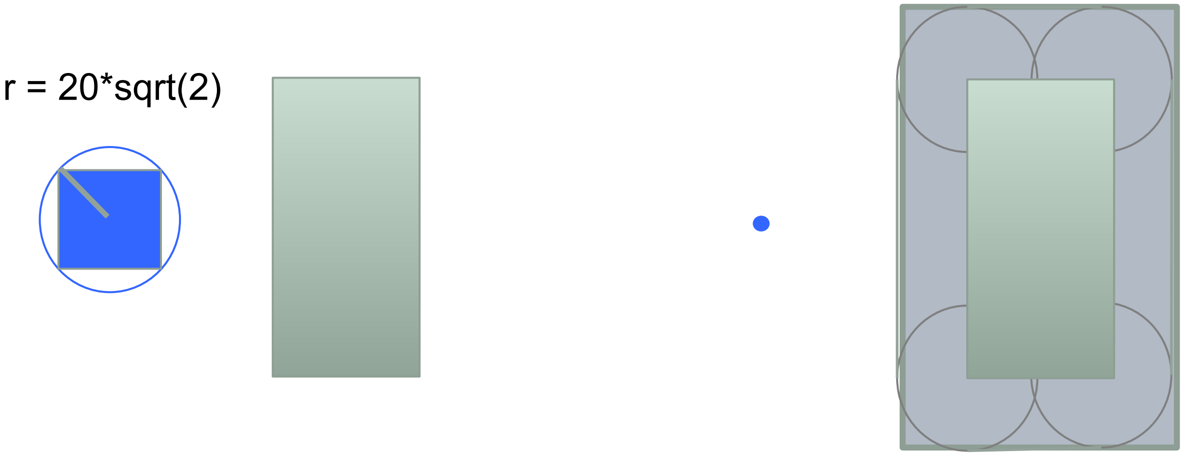
Detected Unexpected Obstacle

Obstacle generates repulsive force

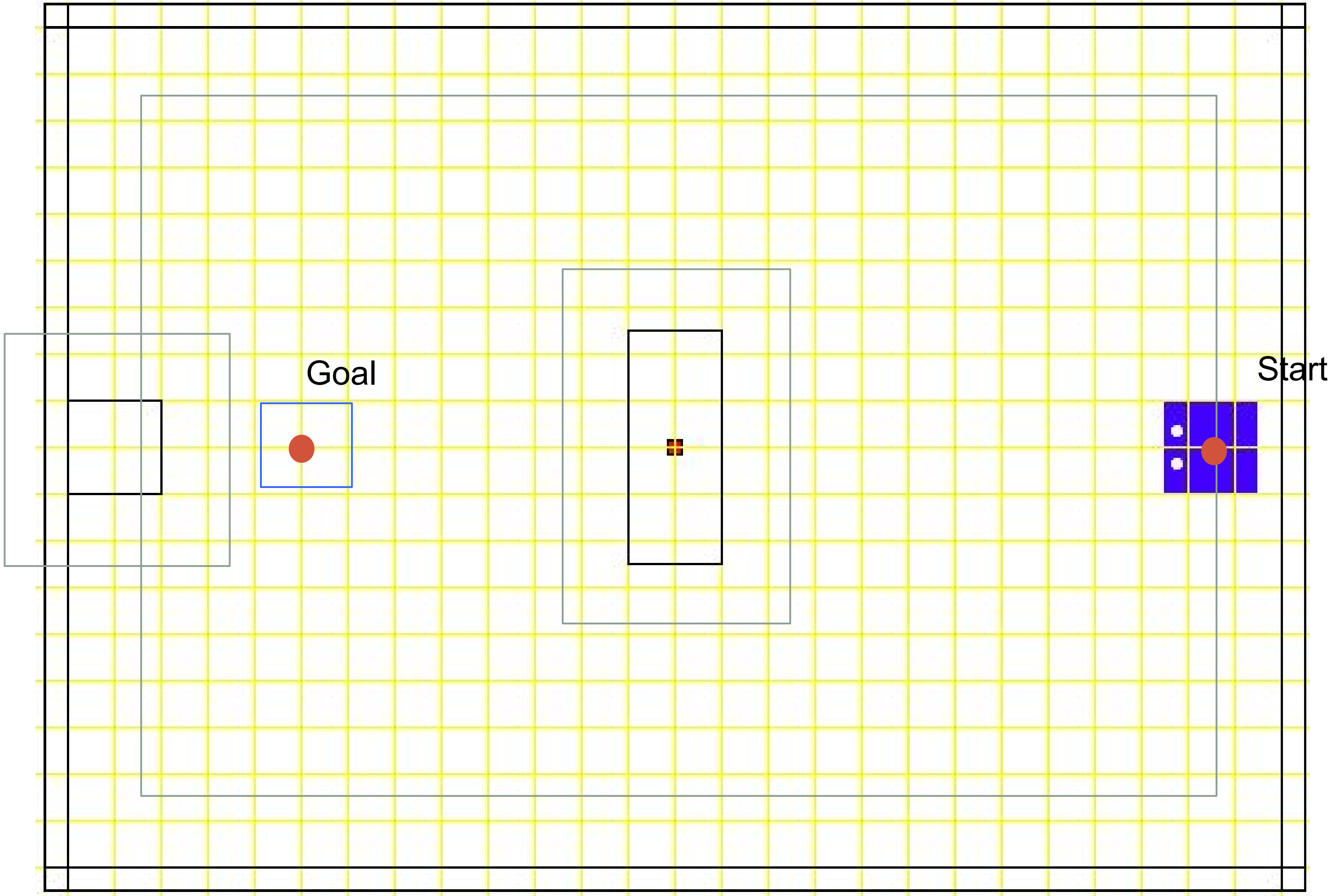
Goal generates attractive force

Simplify Hamster’s Simple World

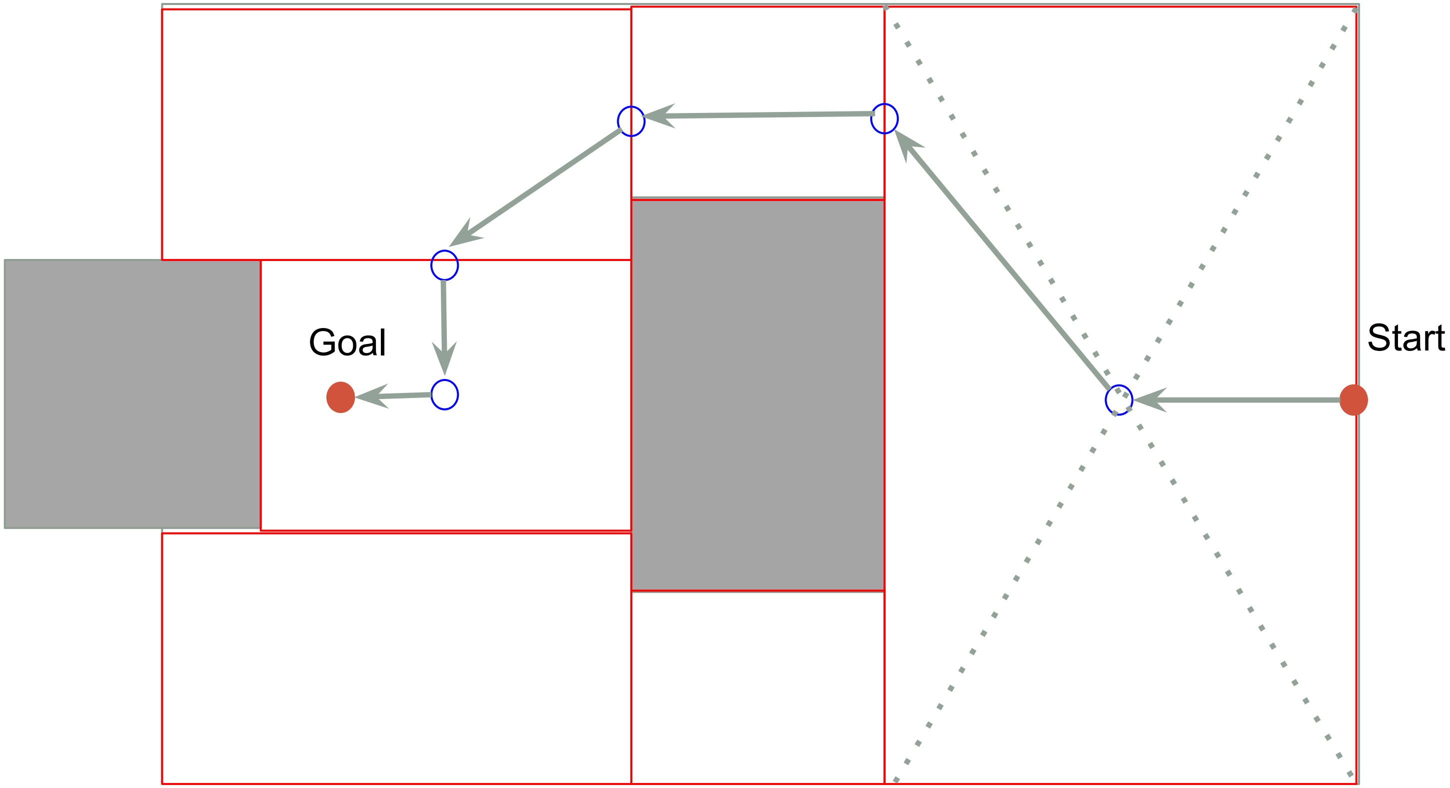
- We approximate Hamster as its Circumscribing Circle (we assume Hamster is a 40mm x 40 mm Square)
- Approximate the C-space obstacles by their bounding rectangle



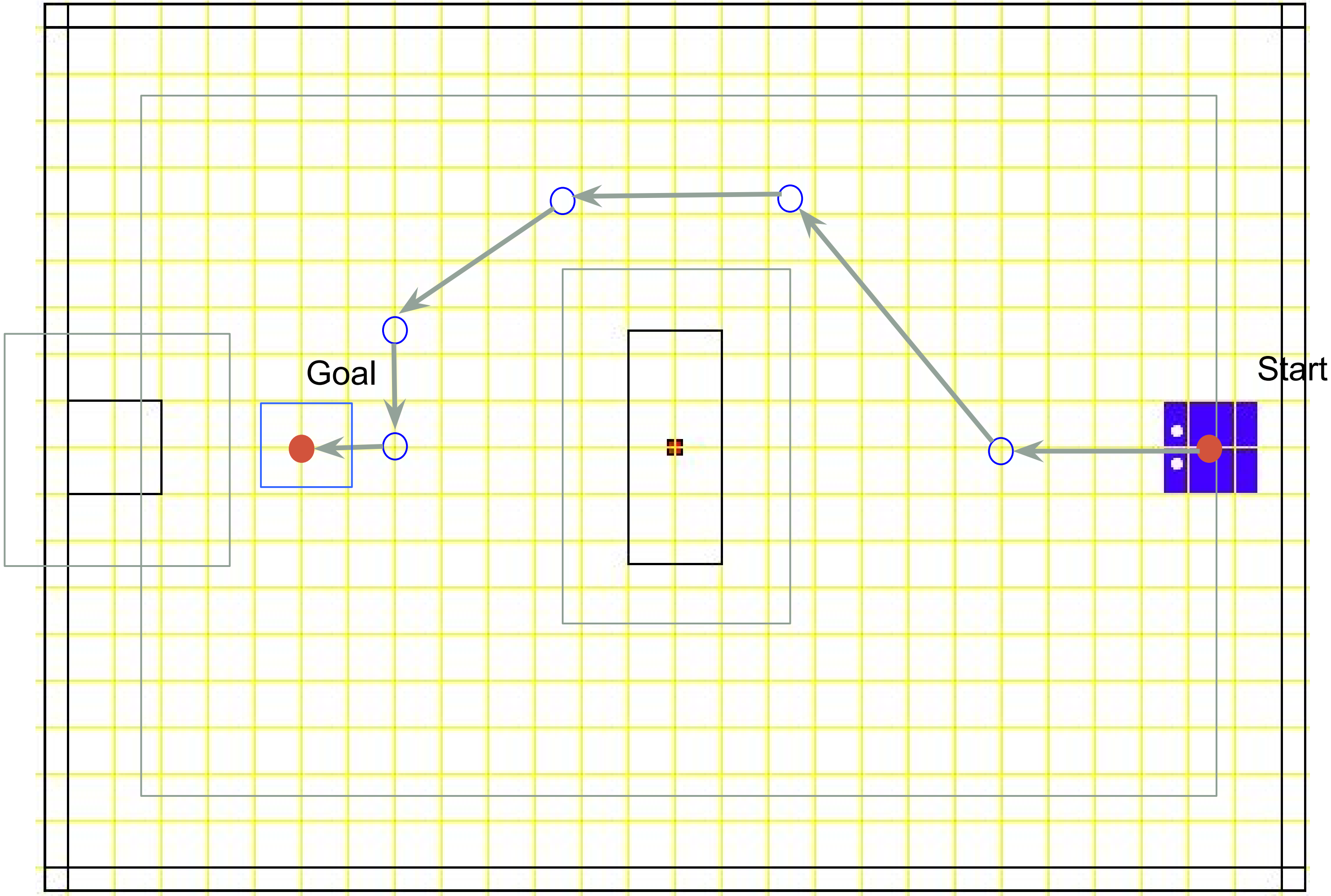
A Simple Work Space / C-space



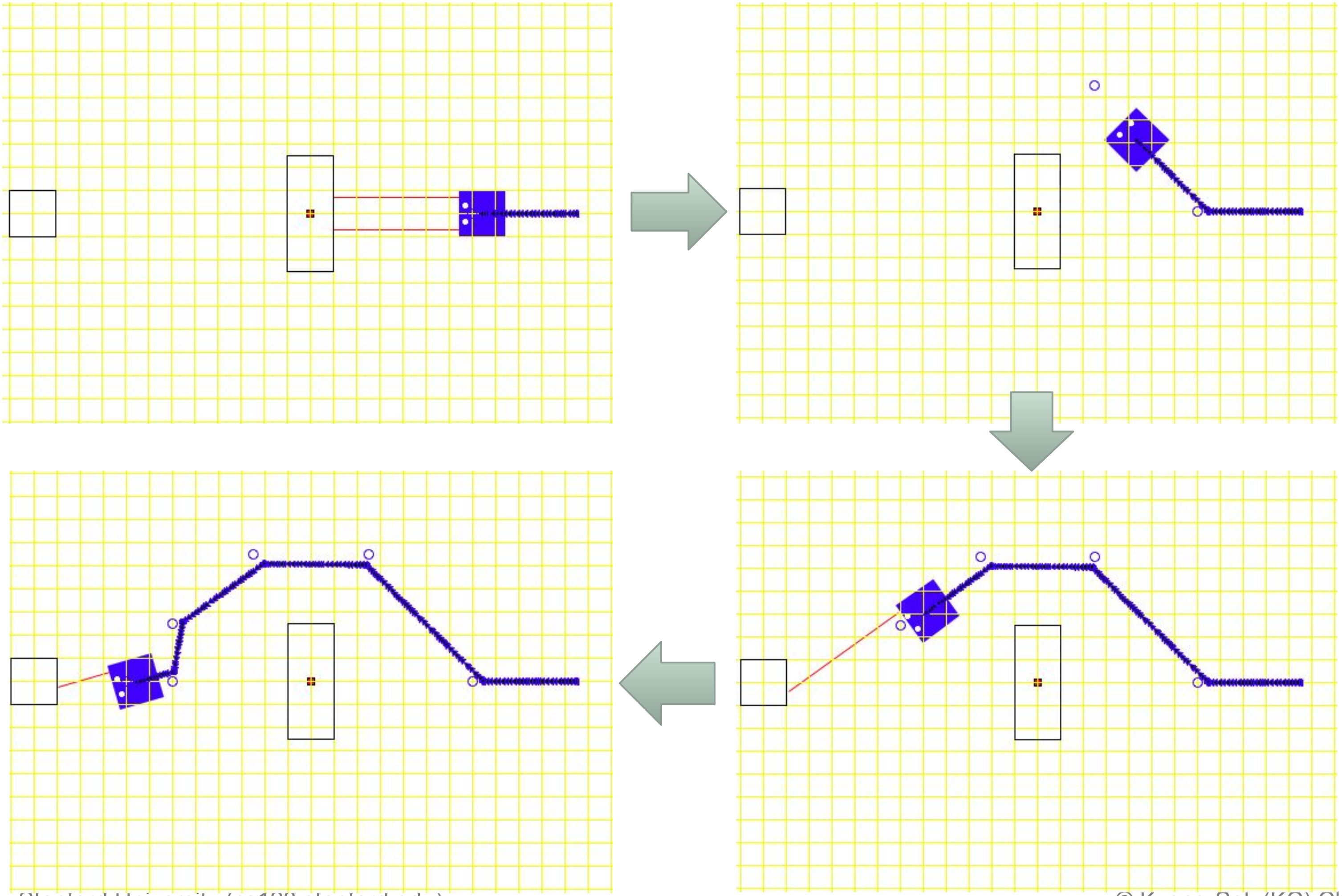
Simple Motion Plan For Hamster Using Exact Cell Decomposition



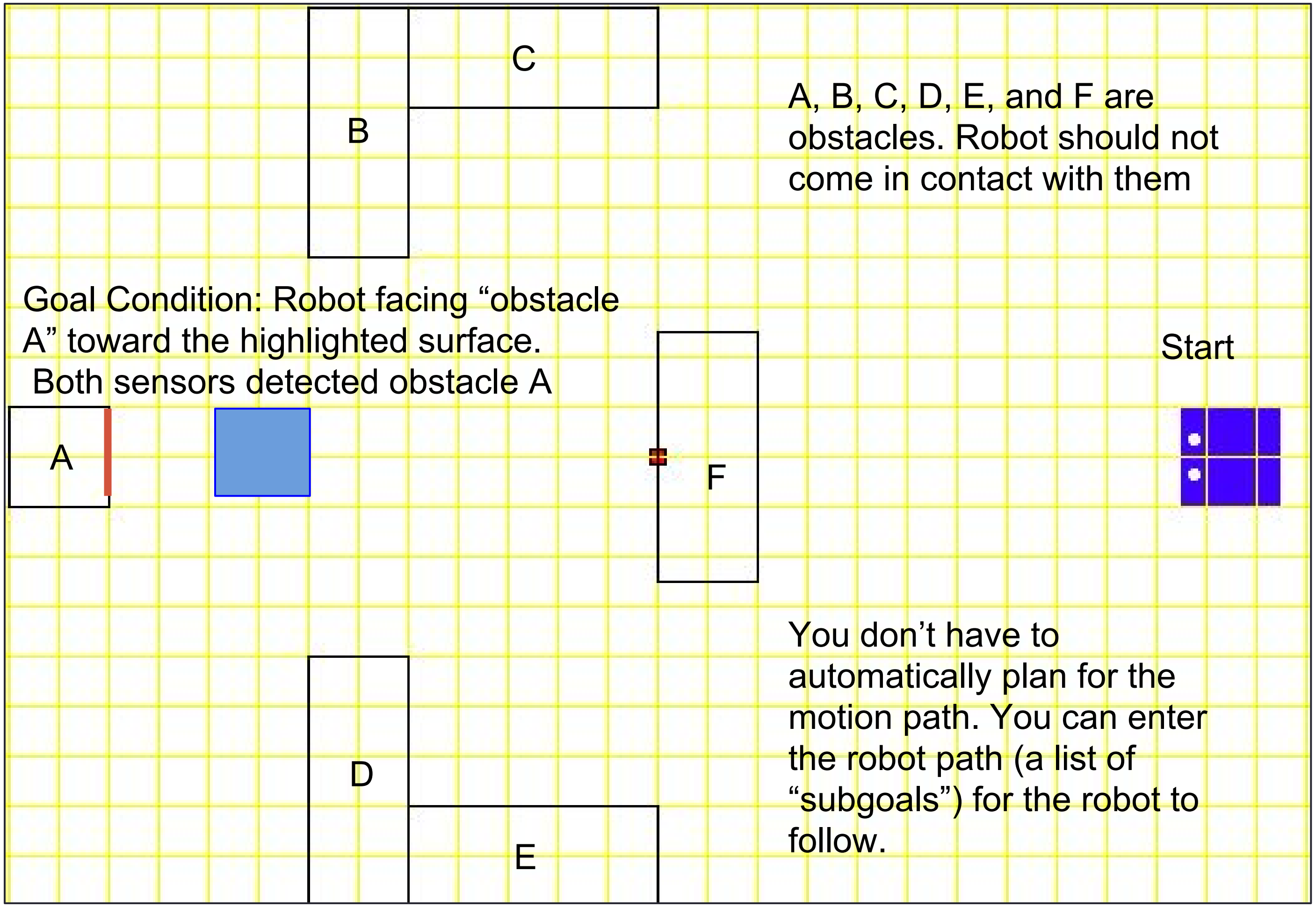
Path in Work Space



Plan Execution In A Perfect (Virtual) World



Homework Part #3-2

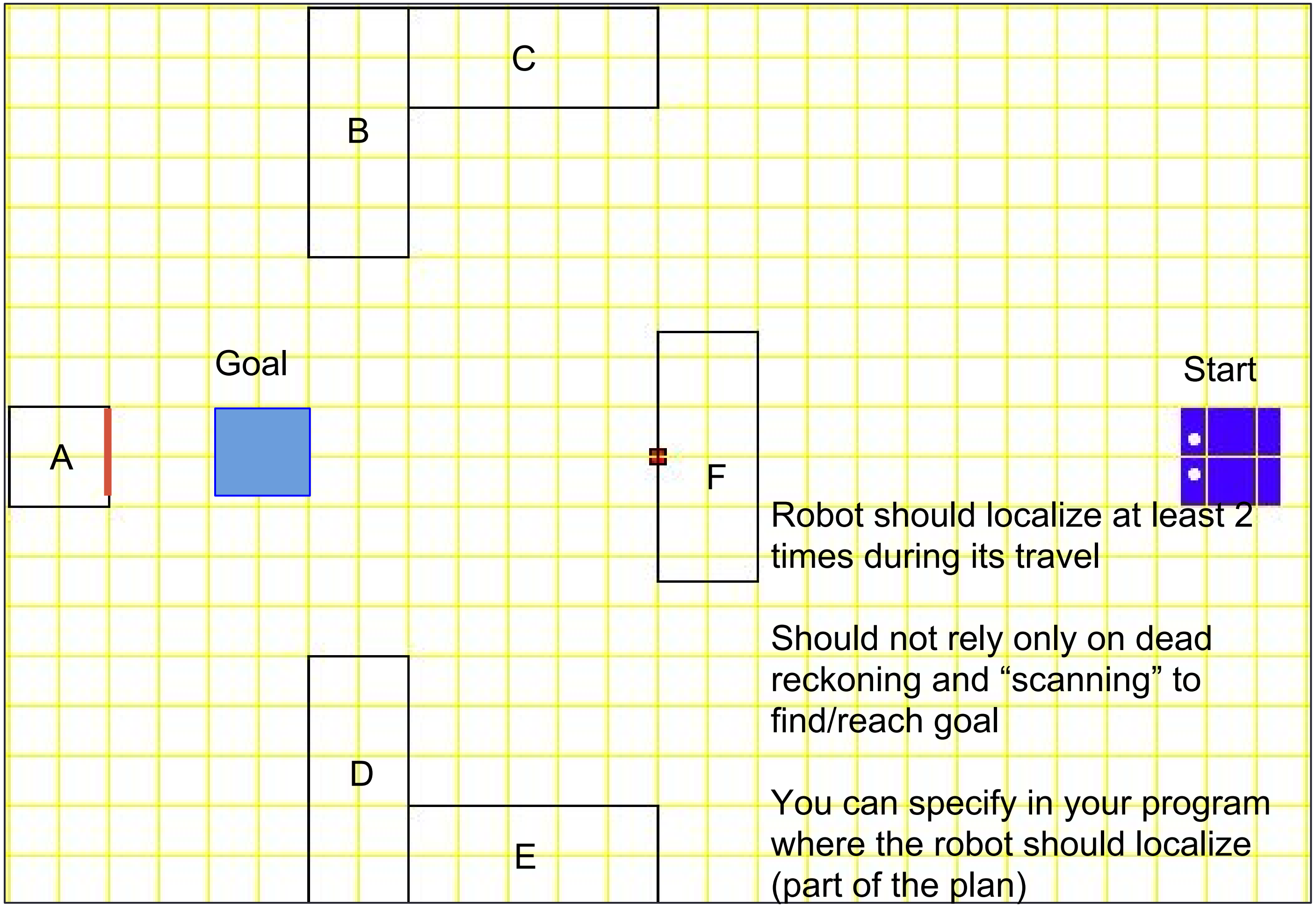


A, B, C, D, E, and F are obstacles. Robot should not come in contact with them

Goal Condition: Robot facing "obstacle A" toward the highlighted surface.
Both sensors detected obstacle A

You don't have to automatically plan for the motion path. You can enter the robot path (a list of "subgoals") for the robot to follow.

Homework Part #3-2



Robot should localize at least 2 times during its travel

Should not rely only on dead reckoning and “scanning” to find/reach goal

You can specify in your program where the robot should localize (part of the plan)

Lec#12: Motion Planning & Control

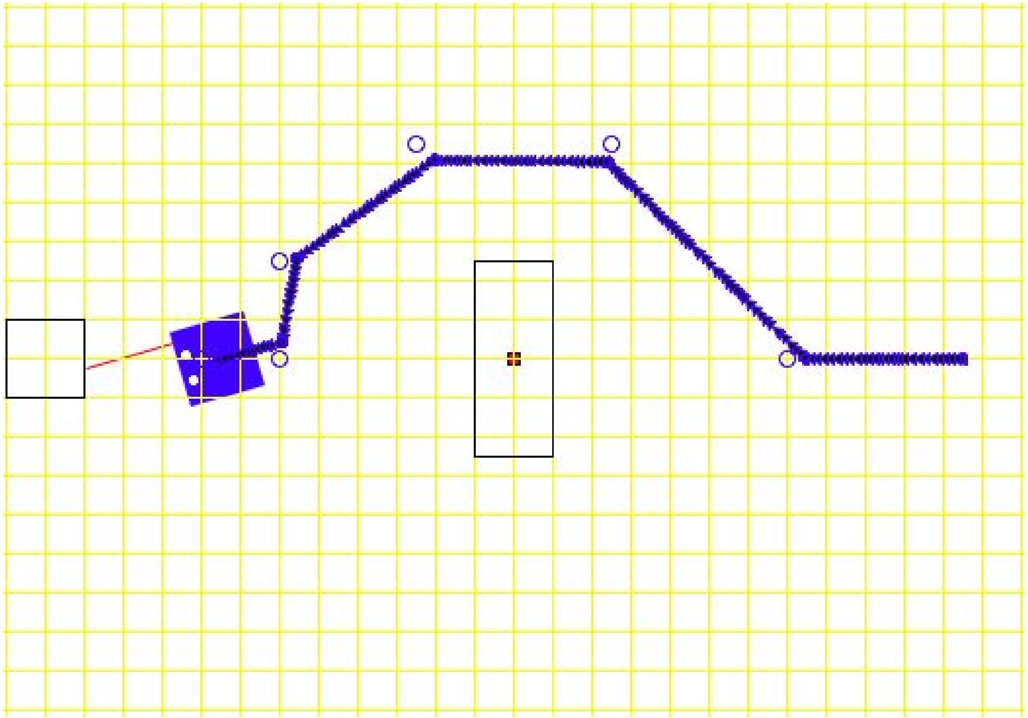
- More on Motion Planning
  - Search (A*)
    - Uninformed (Blind): BFS, DFS
    - Informed (Heuristic): Evaluation function: Dijkstra's, A*
  - Potential Field Method
- More on Control Under Uncertainty
  - Motion "Primitives"
  - Avoiding "Unexpected" Obstacles
- More on Assignment#3-2
  - student demo (Starbuck reward still good)

“General” Controller for Hamster

- Separating Planning and Control
  - Should not hard-code the controller together with the planner
  - The planner outputs a list of “sub-goals”
  - The controller translates the sub-goal list into a sequence of executable “motion primitives”

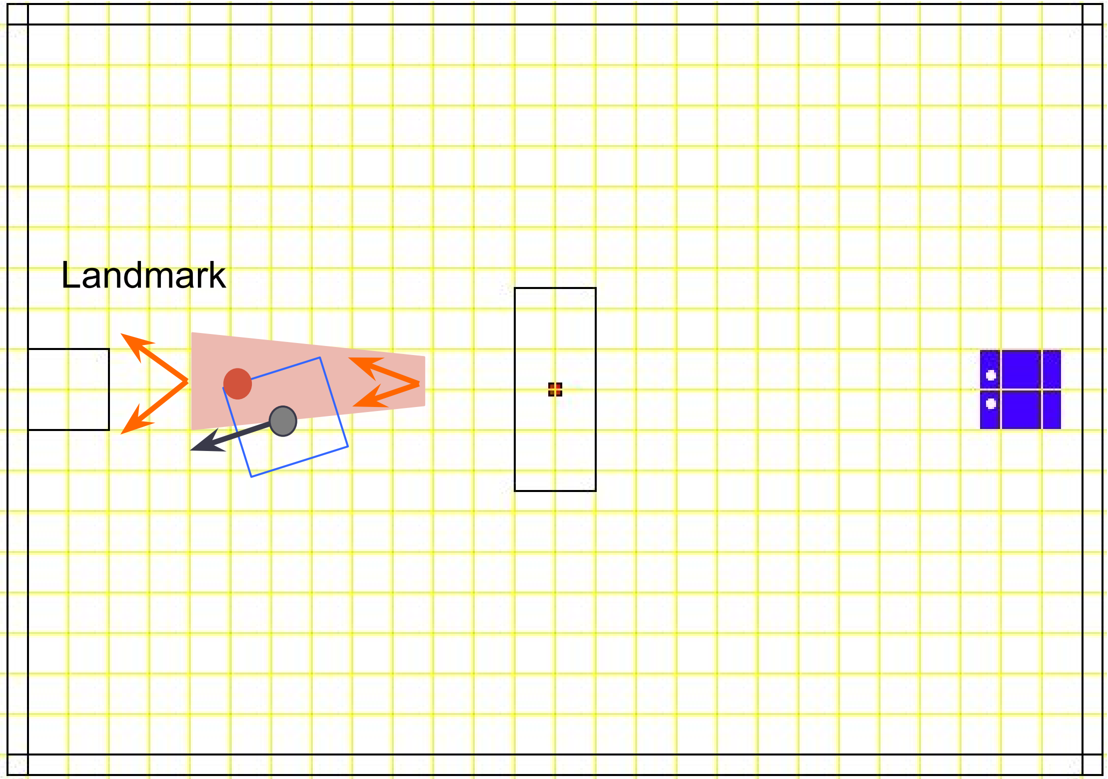
Motion Control: Motion Primitives

- Perfect World:
  - Move to (x, y, a)
  - Terminate when getting close enough to (x, y, a)



Motion Primitive: Control Uncertainty

- Real World – Control Uncertainty
  - Move along d (direction)
  - Terminate with some sensor



Final Project

- Mobile Robot Programming
  - Event driven programming: FSM
  - Navigation
    - modeling: hamster (sensor, effector), environment
    - localization: local (IR, floor), global (landmark), vision
    - planning: c-space, cell decomposition, search
      - local (reactive), global
    - execution: motion primitives, completion (fail, success)
    - UI/UX: graphics, keyboard, sound, LED, motion, etc
  - Creativity: fun factor
  - Team of 2+ people with 2+ robots
  - 5 min oral presentation + 10 min demo: attendance (full 2 hours)
- Project should be well defined
  - Clear objectives (goals), gameplay, completion (win/loss, success/fail)
  - Precise definition of “initial state”, “final state” and “transition”
  - Assumptions: environment, human intervene, moving objects, etc
- CS 123 Final Project Proposal Guidelines

My Final

Dave’s run
Mammoth Mt. Ski Resort
CA USA 2014