ROB550 Robotic Systems Lab

Winter 2018

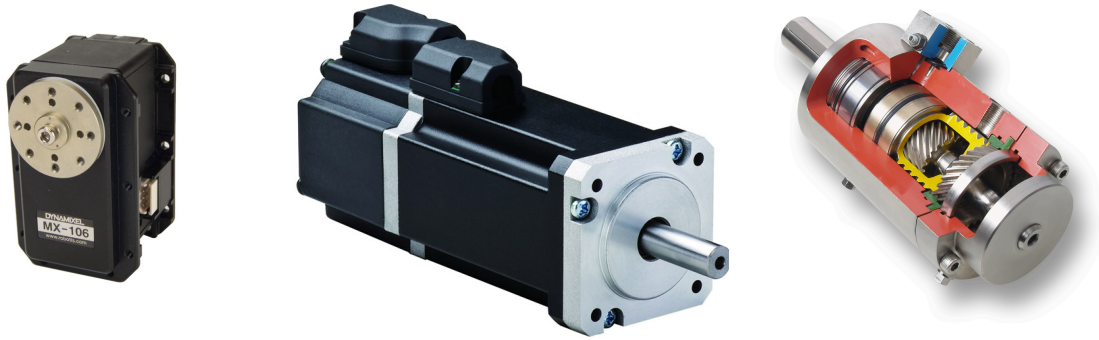
Introduction to Robotic Manipulation

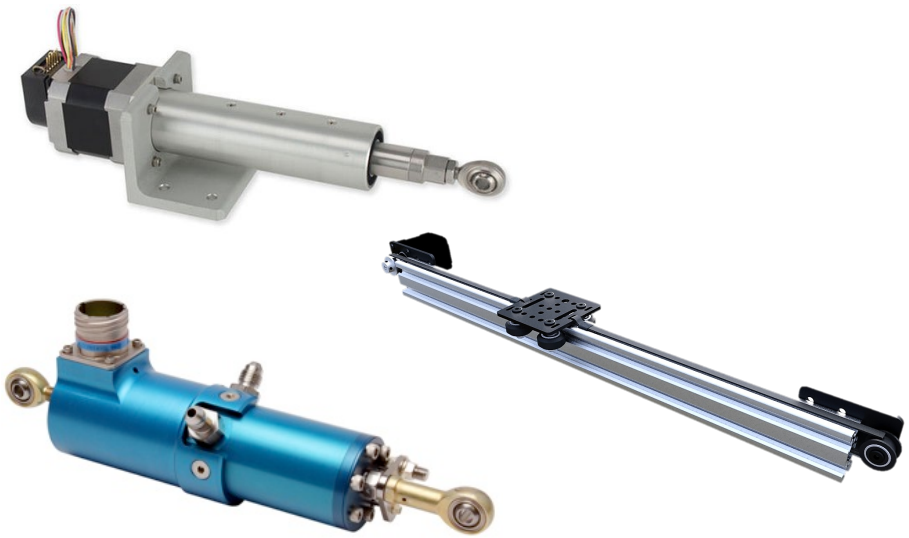
Lecture #1

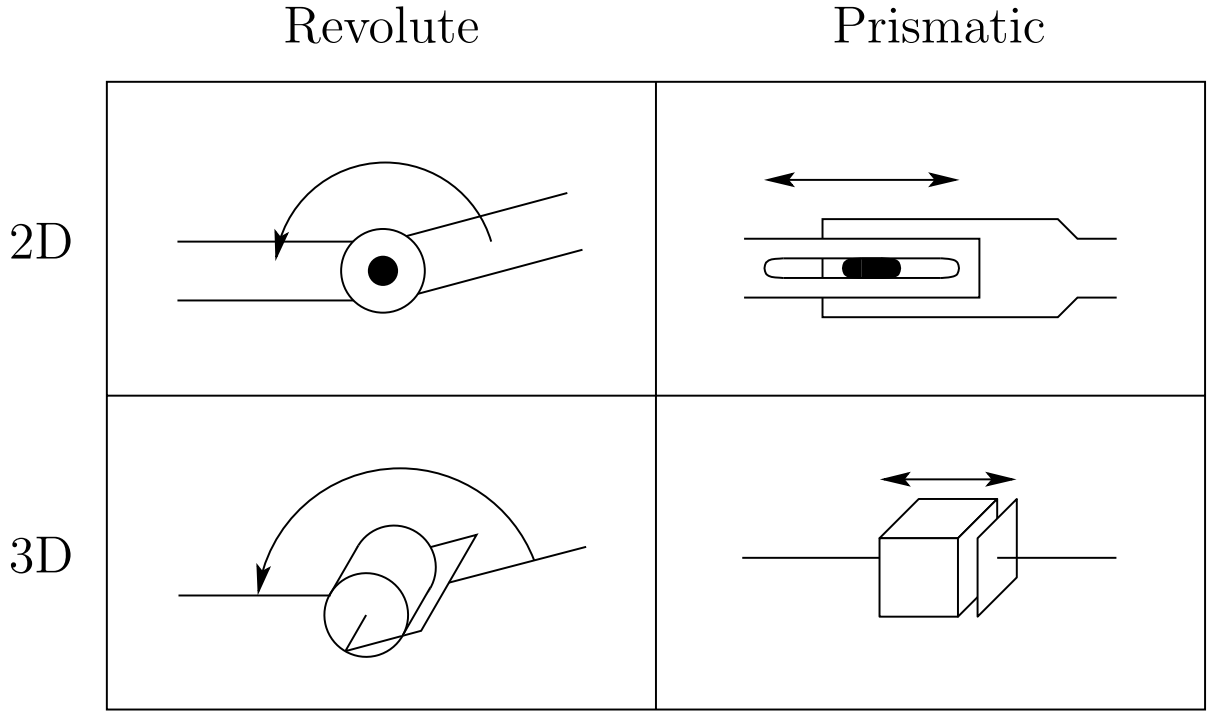
Spong CH1

Joints

- Two types of actuated joints
- Single degree of freedom (DOF)
- Rotational → Revolute
- Linear → Prismatic

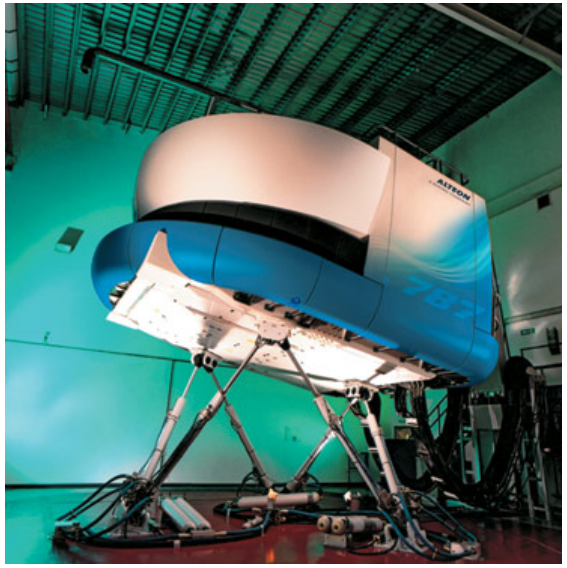






Configurations

- Serial: linear kinematic chain
- Parallel: two or more kinematic chains in parallel



Serial Kinematic Chains

- Serial manipulators composed of Links actuated by Joints
- Axis of rotation or translation of joint i is z_i .
- Configuration is the complete specification of the location of every point on the manipulator.
- Configuration is represented as a vector $\boldsymbol{q}$ of joint variables
- $\boldsymbol{q} = [q_1, q_2, \ldots q_n]$, q_i is θ_i for revolute and d_i for prismatic joints
- A chain has n degrees of freedom if configuration can be specified by n parameters
- A chain is kinematically redundant with more than 6DOF

Kinematics

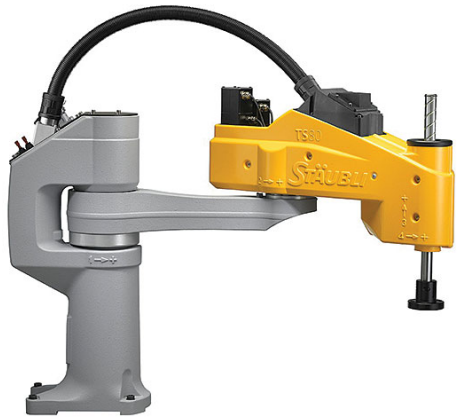
- Describes the motion of points, bodies, and systems of bodies .
- Forward Kinematics: Compute end effector position and orientation $\boldsymbol{x}$ given configuration vector $\boldsymbol{q}$.
- Inverse Kinematics: Compute joint vector $\boldsymbol{q}$ to yield end effector position and orientation $\boldsymbol{x}$ relative to the base frame.
- Forward Kinematics is typically a simple problem for serial manipulators.
- Inverse Kinematics problem becomes difficult for high DOF systems.

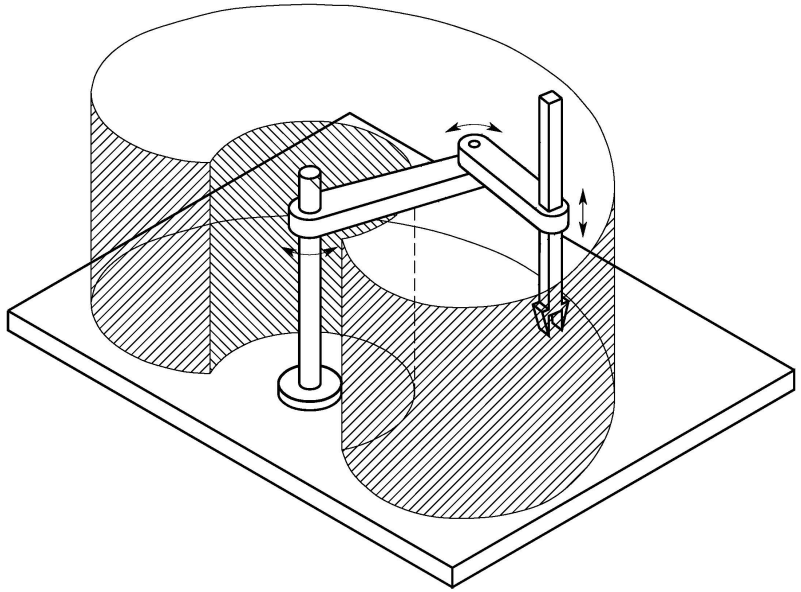
Common Kinematic Chains

- Robotic manipulator often has a base chain (3DOF) and an effector chain (3DOF) to give full 6DOF of end effector
- Articulated → **RRR**
- Spherical → **RRP**
- SCARA → **RRP**
- Cylindrical → **RPP**
- Cartesian → **PPP**

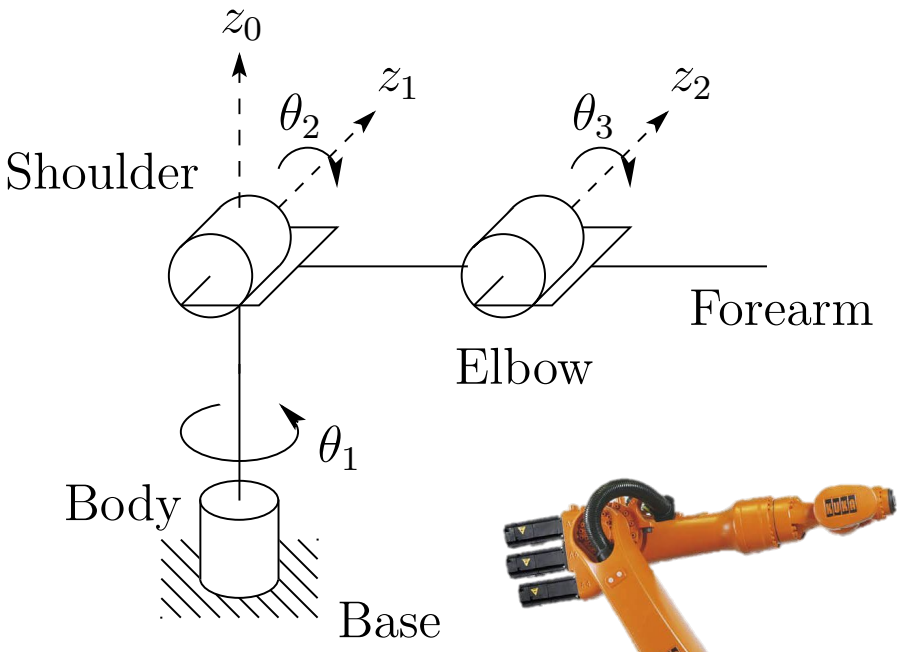
Workspace

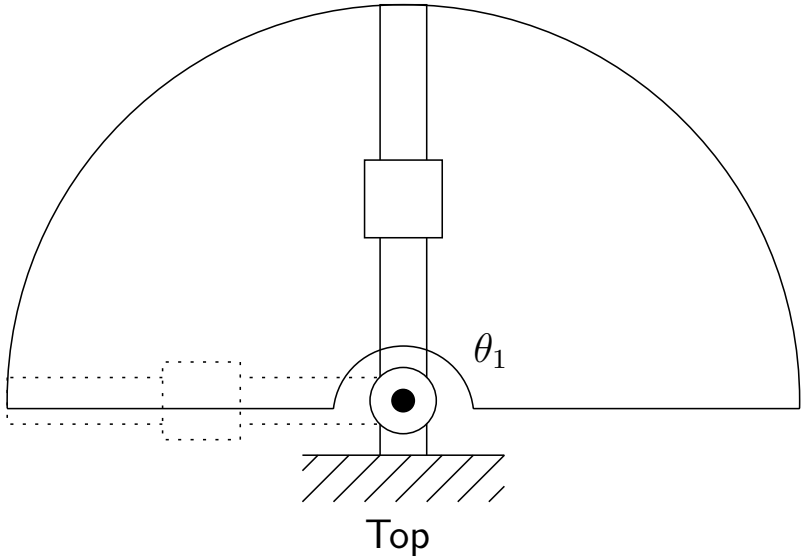
- Total volume swept out by the end effector as the manipulator executes all possible motions
  - Reachable workspace: points that can be reached by the manipulator
  - Dexterous workspace: points that can be reached with arbitrary orientation

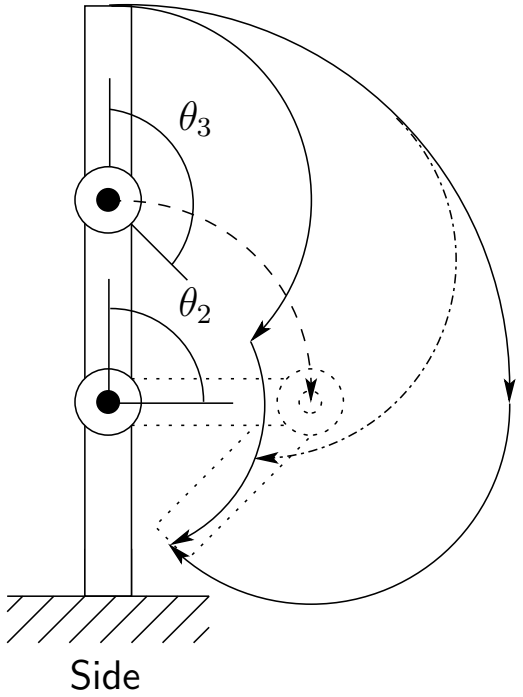




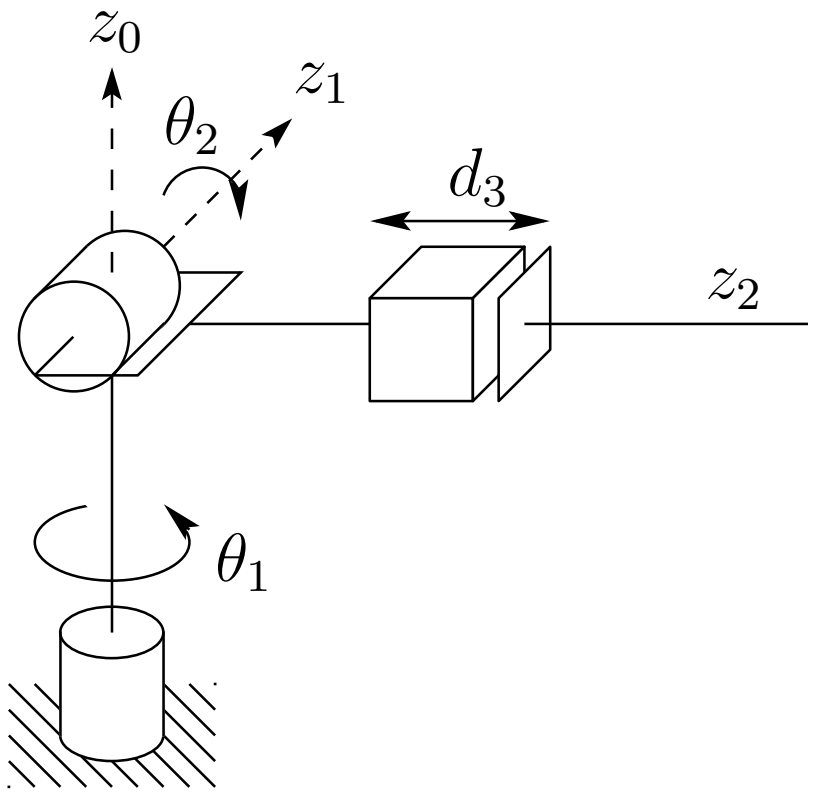
Articulated

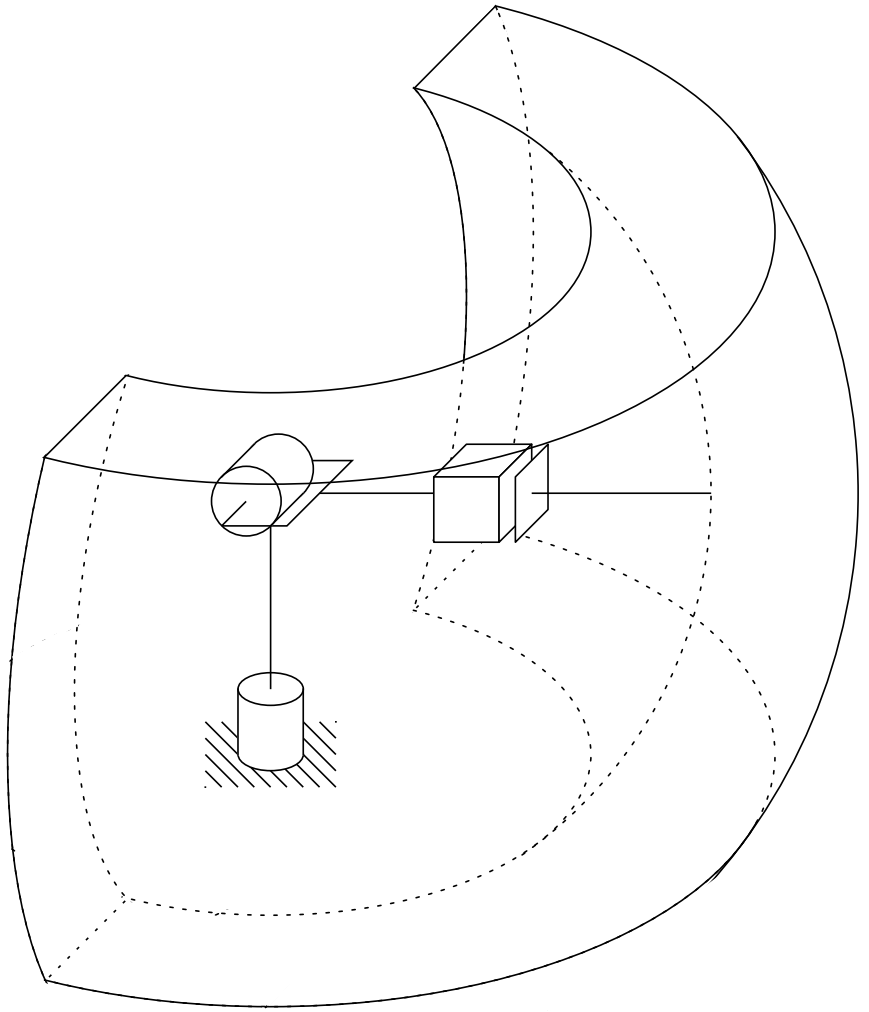




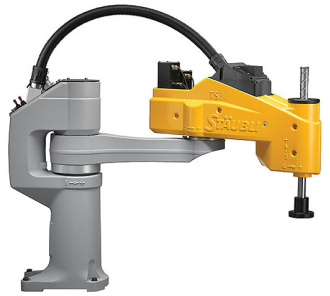


Spherical

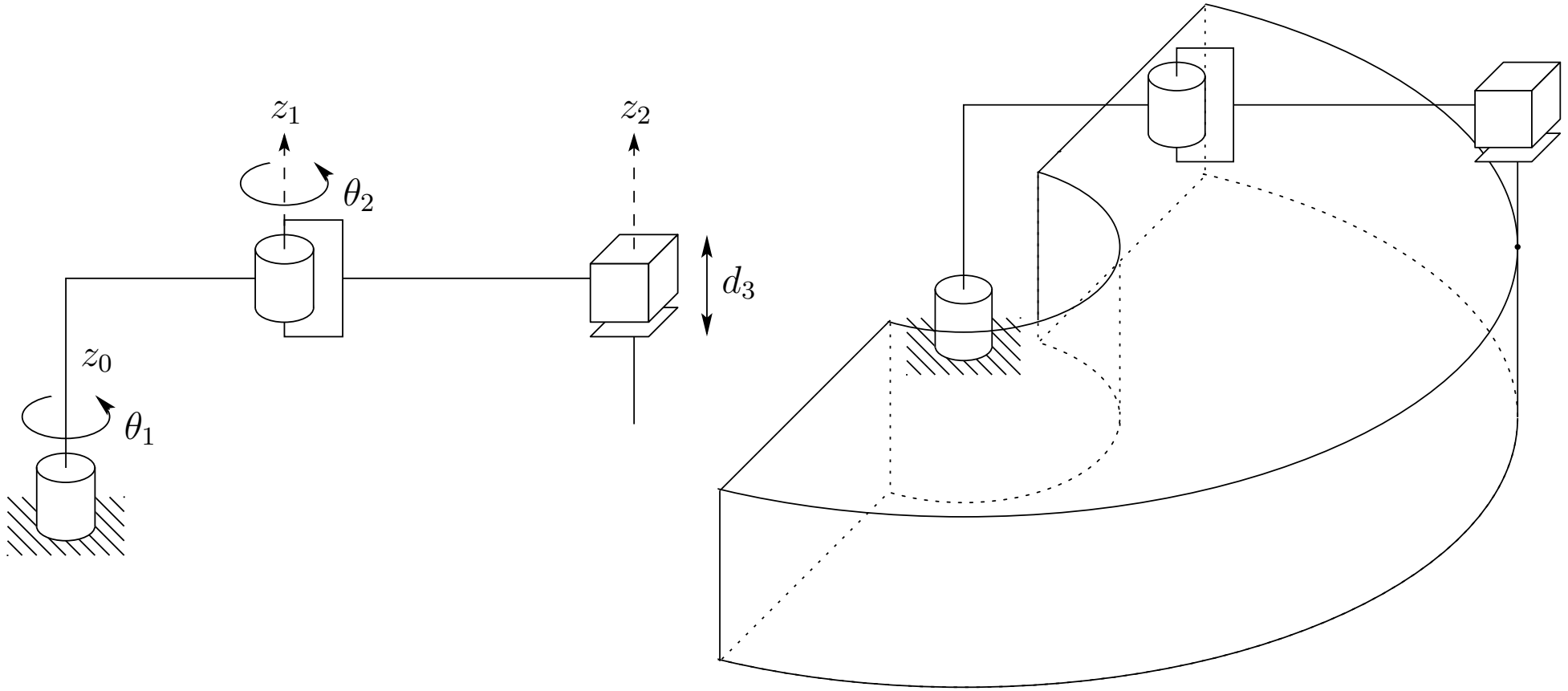




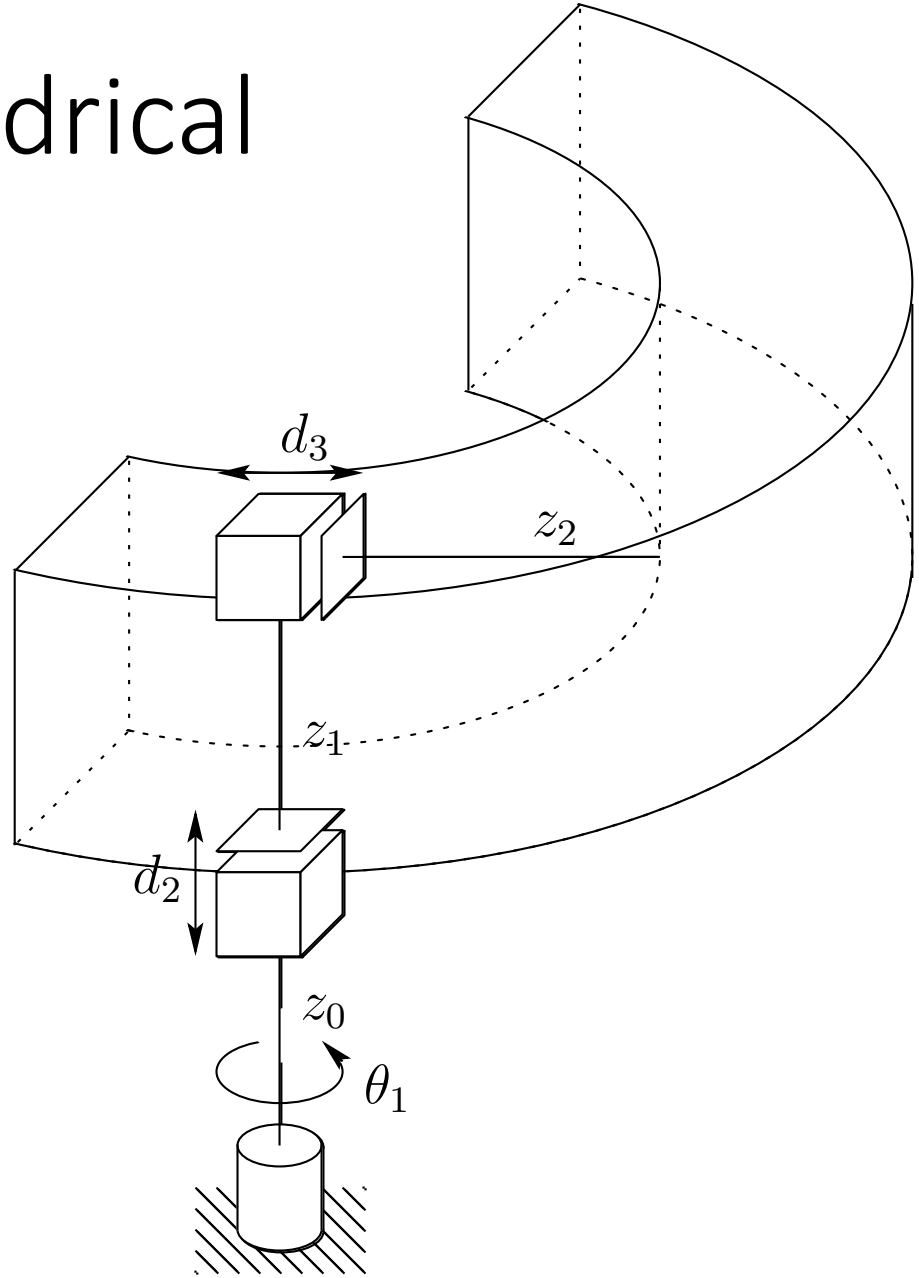
SCARA



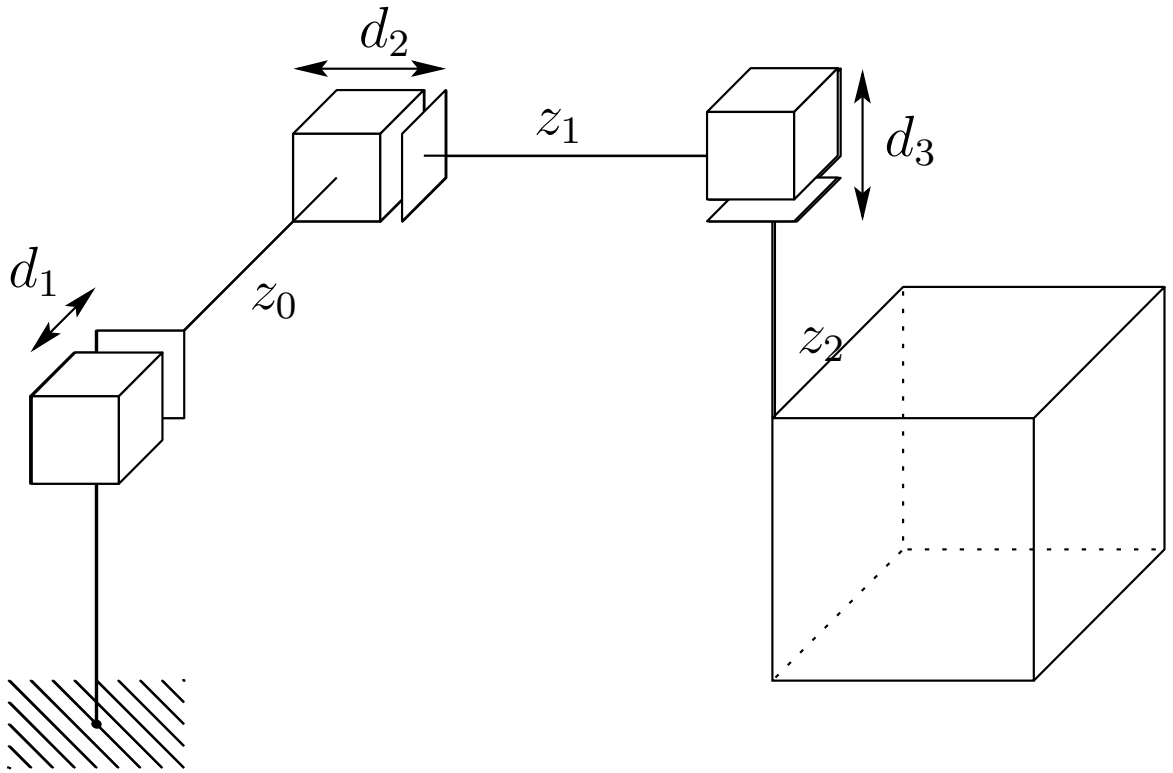
- Selective Compliant Articulated Robot for Assembly



Cylindrical



Cartesian

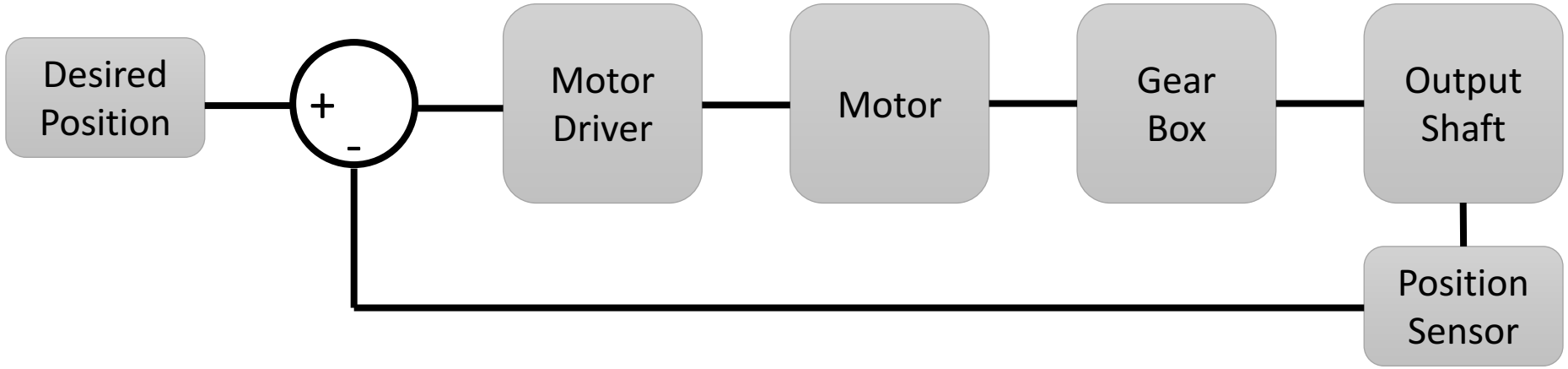


Servo Motors

Lecture #2.1

Servo Motors

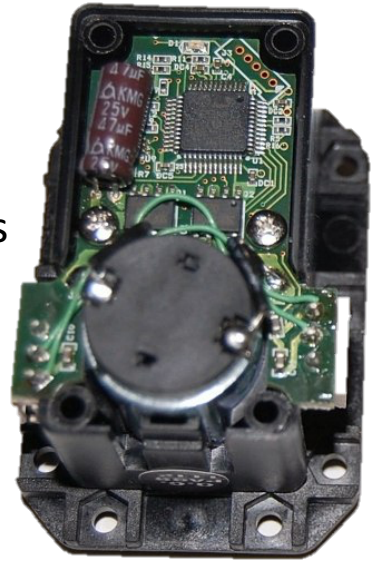
- A servo measures the absolute rotational position of the output shaft
- Uses a feedback loop to control position of the motor
- Advanced servos have more advanced feedback loops to control angular velocity and angular acceleration.



What's inside?

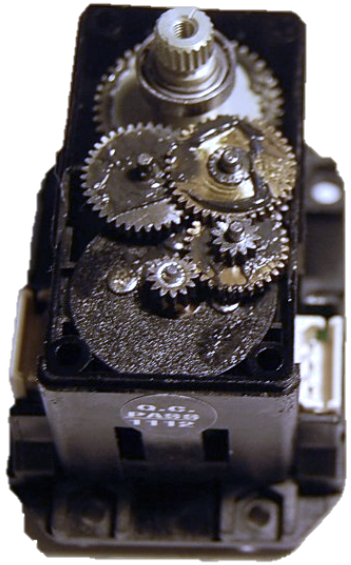
ARM M3 Processor

Motor Drivers



Maxon DC motor

Gear box (193:1)

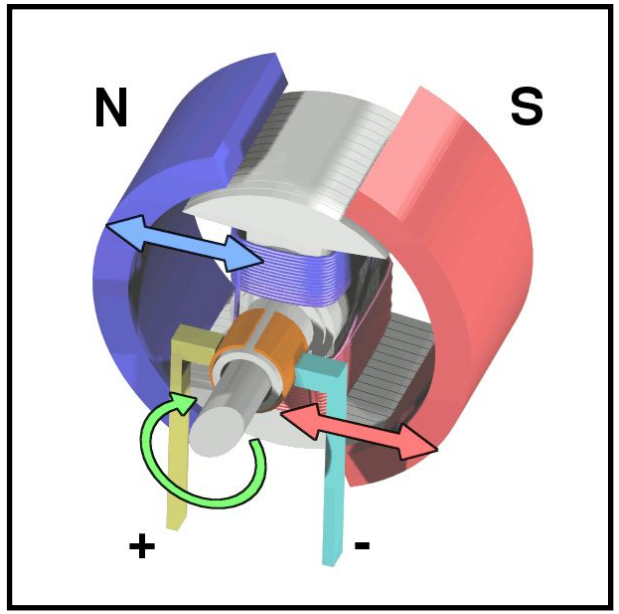


Magnetic Encoder

Magnet

Brushed DC Motor - Basics

- DC current produces magnetic fields in coils
- Magnetic fields align with those of the permanent magnets
- Commutator switches direction of current, and thus polarity of magnetic field to keep motor rotating
- Contact to the commutator made with metal brushes (hence the name)
- As motor spins, field from permanent magnets creates a back EMF in the coils (generator)



Motor Model

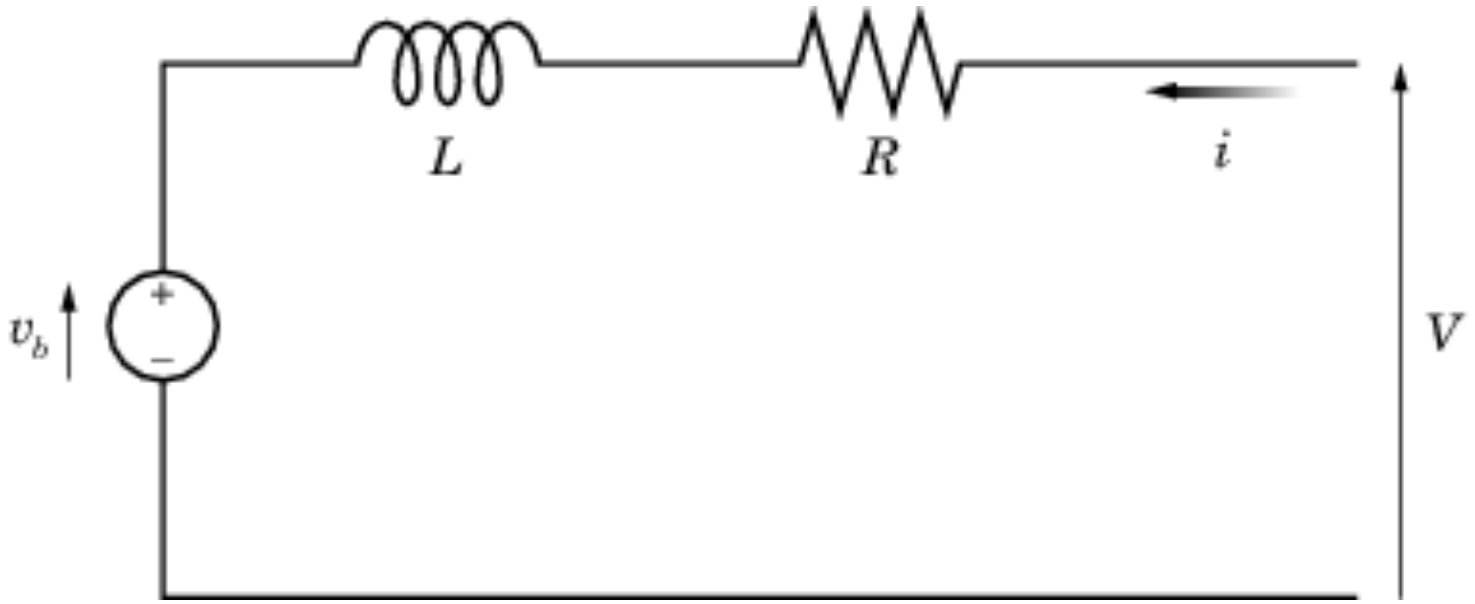
$$v_b = k_v \omega$$

$$\tau = k_\tau i$$

$$\tau \omega = v_b i$$

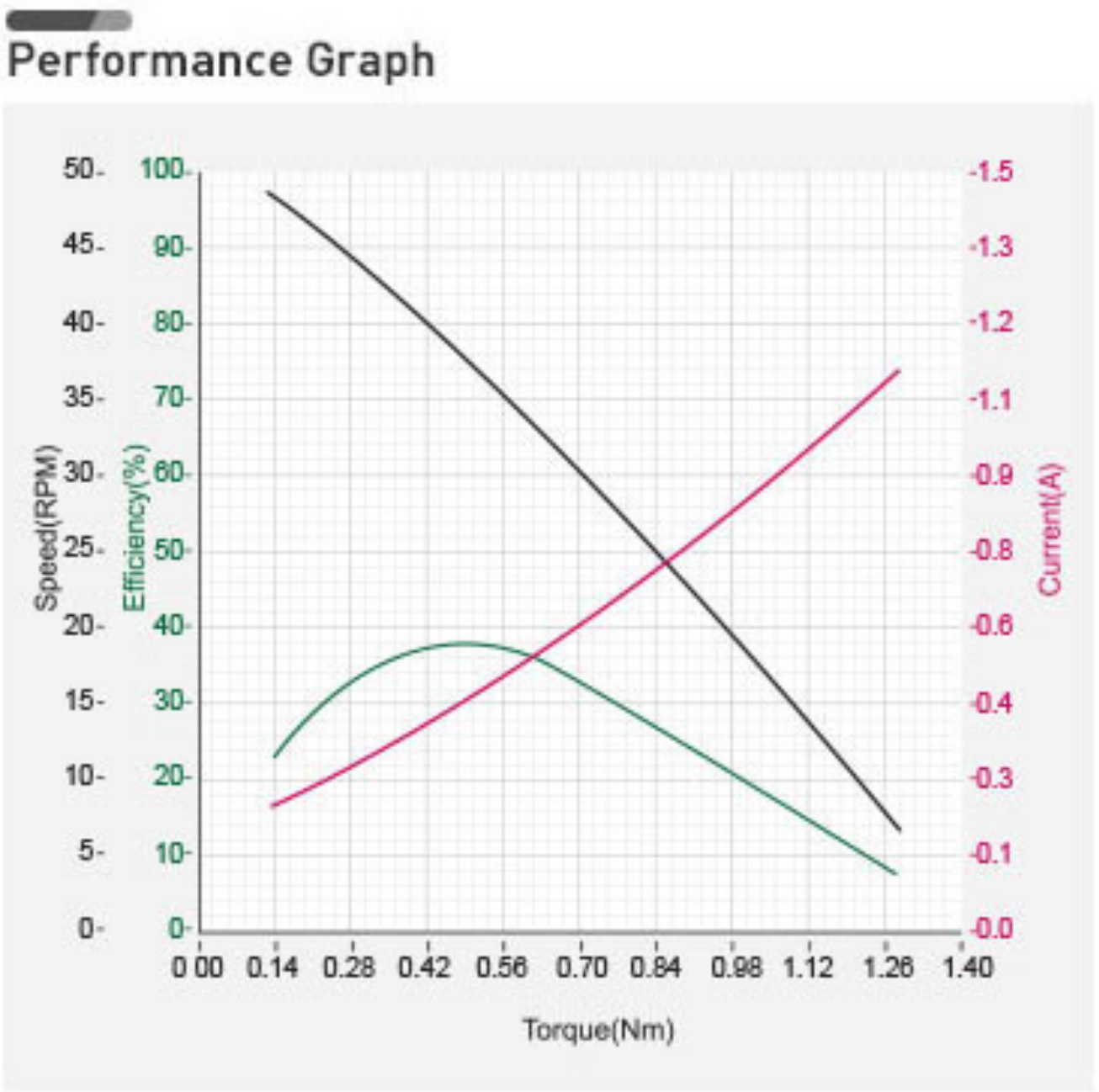
$$i = \frac{V - v_b}{R}$$

In steady state



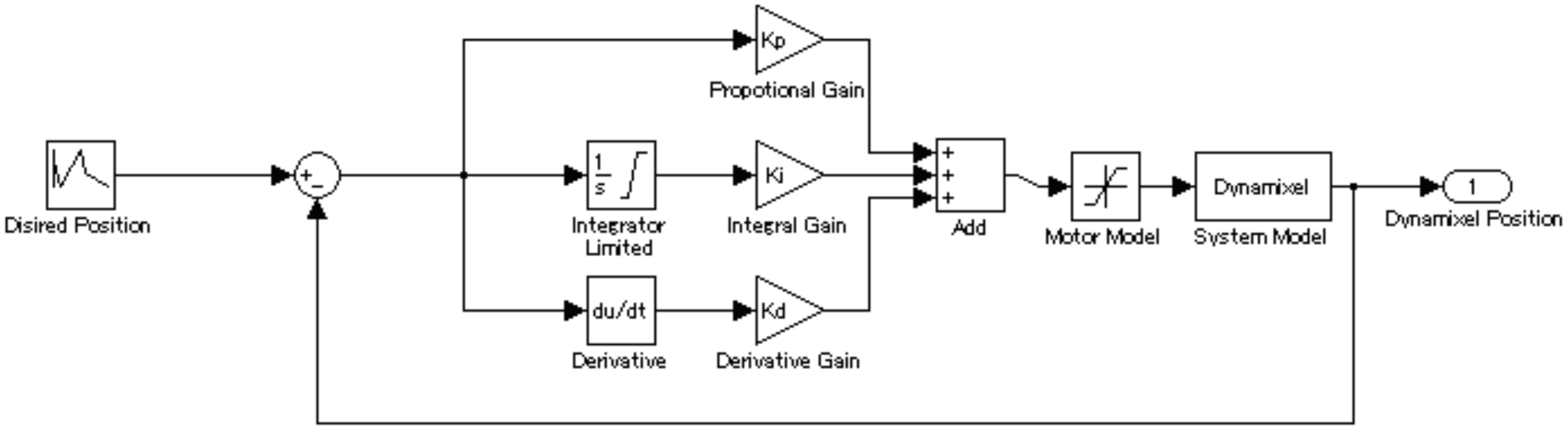
Actual performance

- Motor produces most torque and draws most current at low speed
- Max torque is produced, and max current is drawn when motor is stalled
- Stalling motor could lead to overheating/smoke
- Want to limit torque used to fraction of stall torque



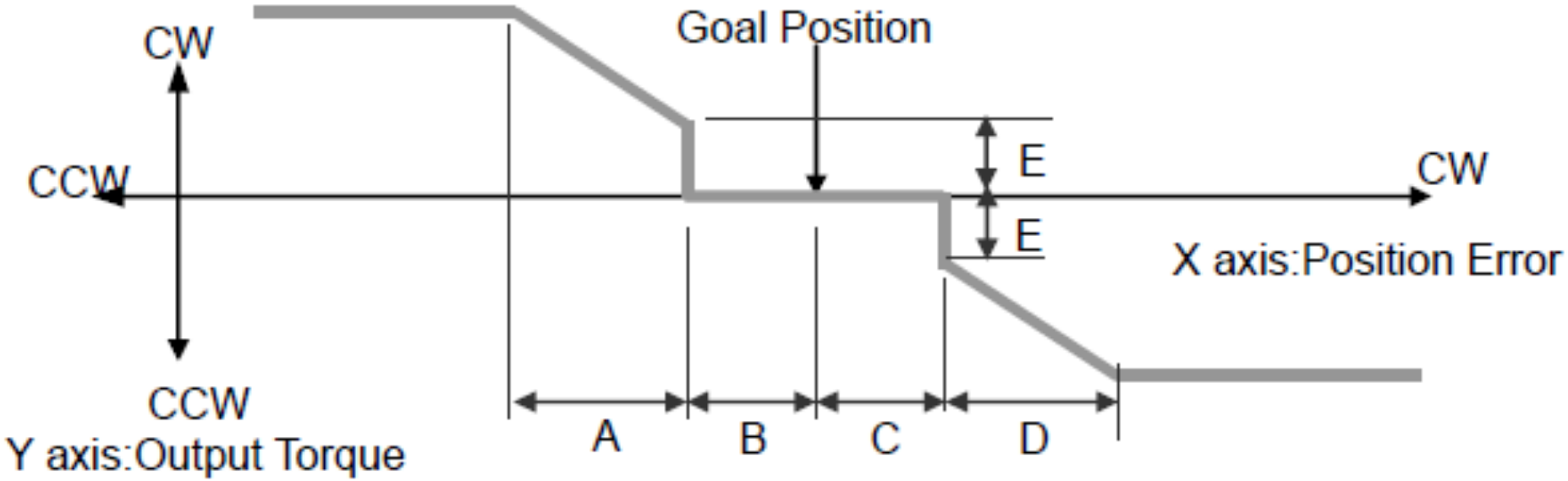
Position Controller XL, MX Series

- Simple PID controller



Position Controller AX Series

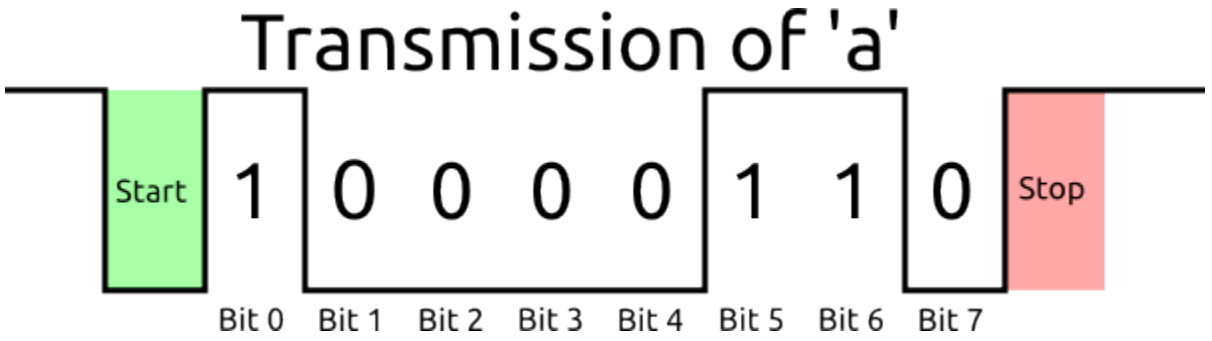
- Proportional Controller with Dead-band & Saturation



A : CCW Compliance Slope(Address0x1D)
B : CCW Compliance Margin(Address0x1B)
C : CW Compliance Margin(Address0x1A)
D : CW Compliance Slope (Address0x1C)
E : Punch(Address0x30,31)

Serial Interface

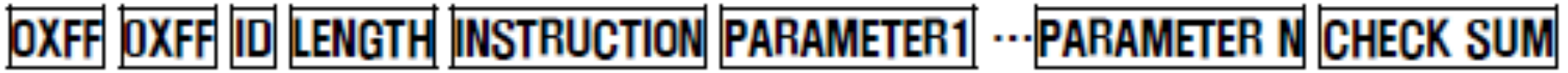
- The Dynamixels uses a standard asynchronous serial protocol
- This protocol is similar to RS-232 and RS-485, but uses a single wire for both transmit (TX) and receive (RX)
- Uses fixed rate of 1Mbaud, 8 bit transmissions, 1 stop bit, no parity bit



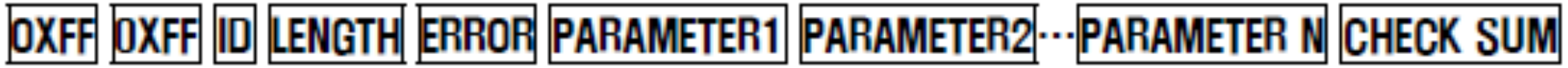
Dynamixel Serial Messages Protocol

1.0

Command



Status



2.0

Command

Header			Reserved	Packet ID	Packet Length		Instruction	Parameter			16bit CRC	
0xFF	0xFF	0xFD	0x00	ID	LEN_L	LEN_H	Instruction	Parameter1	…	ParameterN	CRC_L	CRC_H

Status

Header			Reserved	Packet ID	Packet Length		Instruction	Error	Parameter			16bit CRC	
0xFF	0xFF	0xFD	0x00	ID	LEN_L	LEN_H	0x55	ERROR	Param1	…	ParamN	CRC_L	CRC_H

Dynamixel Driver

- In this lab, we will use dynamixel driver to handle the low level serial interface (so you don’t have to).
- Communication with the driver is handled using LCM.
- LCM is a system for sharing data between different programs/computers using network protocols, specifically UDP Multicast.
- LCM works with and can share data between many different programming languages.
- It is similar (but more efficient) to ROS messages
- There is an LCM tutorial on the google drive.

Python LCM subscription

```
def main():
    lc = lcm.LCM()
    subscription = lc.subscribe("LCM_CHANNEL", lcm_channel_handler)

    try:
        while True:
            lc.handle() #blocks until message recieved

     except KeyboardInterrupt:
            pass



    lc.unsubscribe(subscription)
```

Python LCM publishing

```
def main():
    # initialize LCM
    lc = lcm.LCM()

    # call the constructor for the lcmtype object
    lcm_msg = lcm_msg_t()

    # fill the msg with data
    lcm_msg.data = 1024

    #publish message to lcm
    lc.publish("LCM_CHANNEL", lcm_msg.encode())
```

Command Message

```
struct dynamixel_command_t
{
 int64_t utime;

 // position of servo in radians, [-pi, pi]. A command of zero
 // corresponds to the mid-way position (indicated on the
 // servo by a position notch.)
 double position_radians;
 double speed; // how fast to move to new position [0, 1]
 double max_torque; // torque limit [0, 1]
}
```

Feedback Message

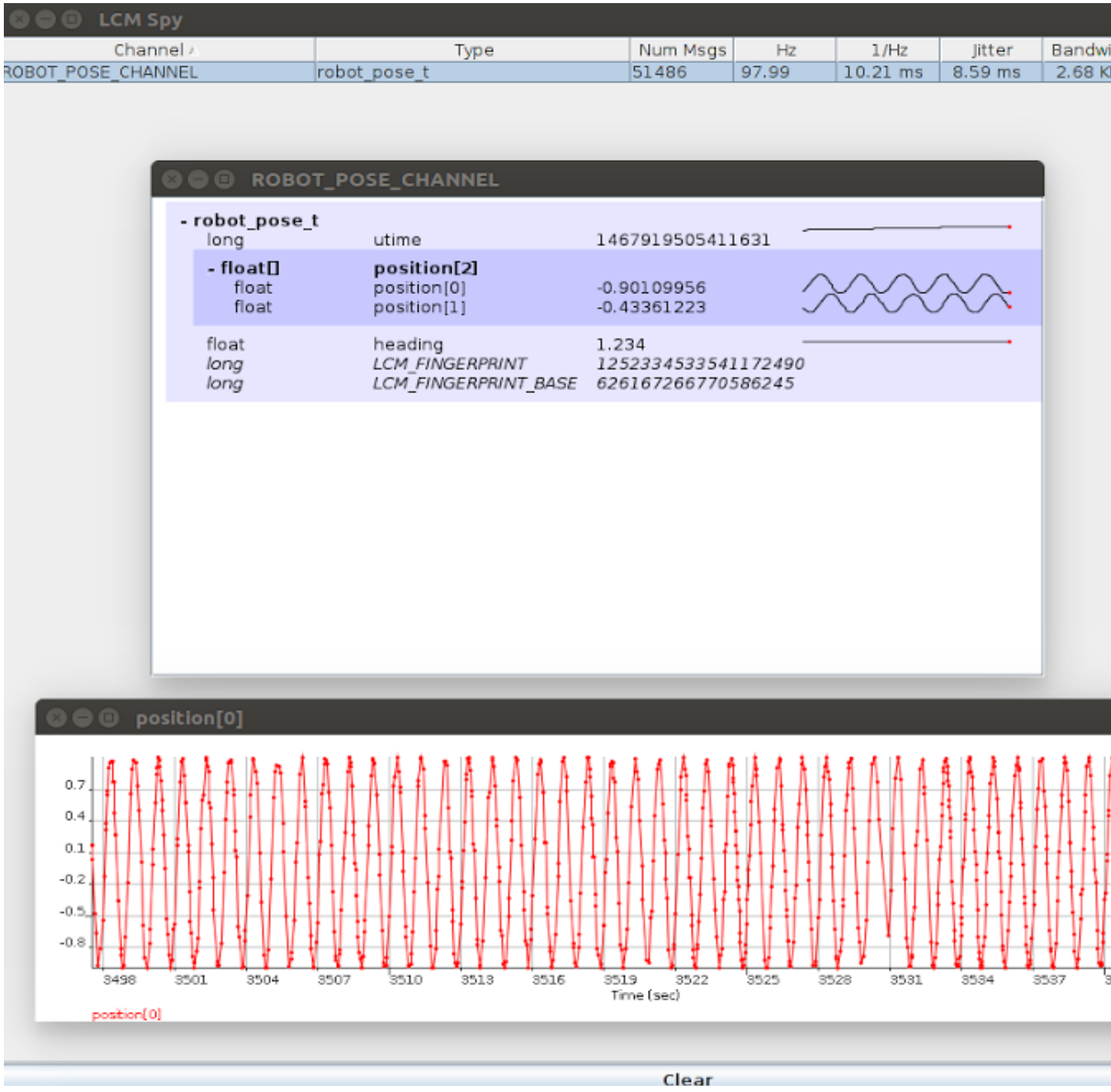
```
struct dynamixel_status_t
{
 const int32_t ERROR_VOLTAGE = 1, ERROR_ANGLE_LIMIT = 2,
    ERROR_OVERHEAT = 4, ERROR_OVERLOAD = 32;

 int64_t utime;

 int32_t error_flags;

 double position_radians; // position of servo in radians, [0, 5.2333]
 double speed; // unsigned measure of angular velocity, [0, 1]
 double load; // measure of motor current, [-1, 1]
 double voltage; // supply voltage (volts)
 double temperature; // temperature (celsius)
}
```

Configuration Messages

```
struct dynamixel_config_t
{
 int64_t utime;
 // these are for MX and XL motors
 byte kp; //0-255 default: 32
 byte ki; //0-255 default: 0
 byte kd; //0-255 default: 0
 //these are for AX motors
 byte compl_margin;
 byte compl_slope;
 //this is for all motors
 byte LED; //0,1 for AX/MX to turn on and off, 0b111 for XL RGB LED
}
```

LCM Spy

- Java application to probe LCM traffic
- Must know location of java archive of lcmtypes
- Java lcmtypes auto generated in our repos with Makefile
- $> source setenv.sh



LCM Logs

- You can record LCM traffic with lcm-logger
- You can playback LCM traffic with lcm-logplayer-gui
- You can write a simple program to decode log files to data files for plotting with matplotlib, Matlab etc.

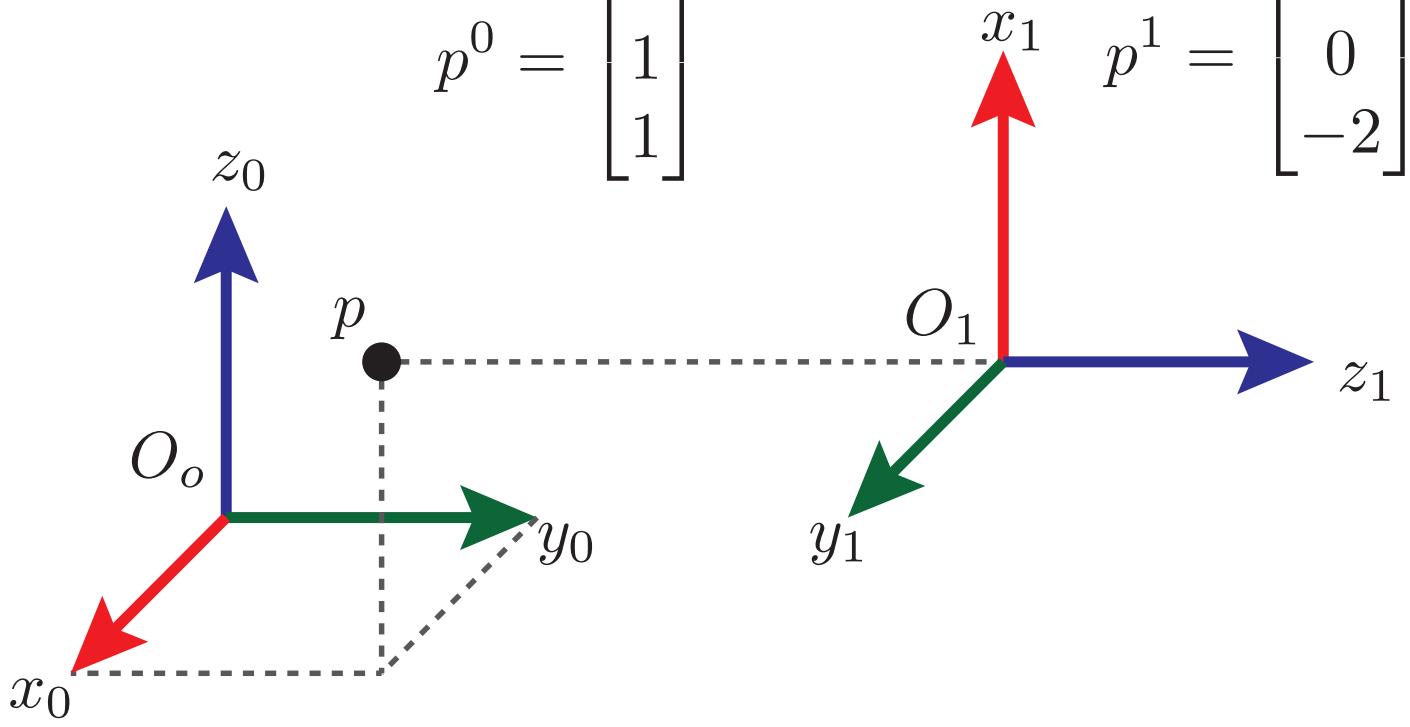
Rigid Body & Other Transformations

Lecture #2.2
Spong CH2

Points & Frames

$$p^0 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

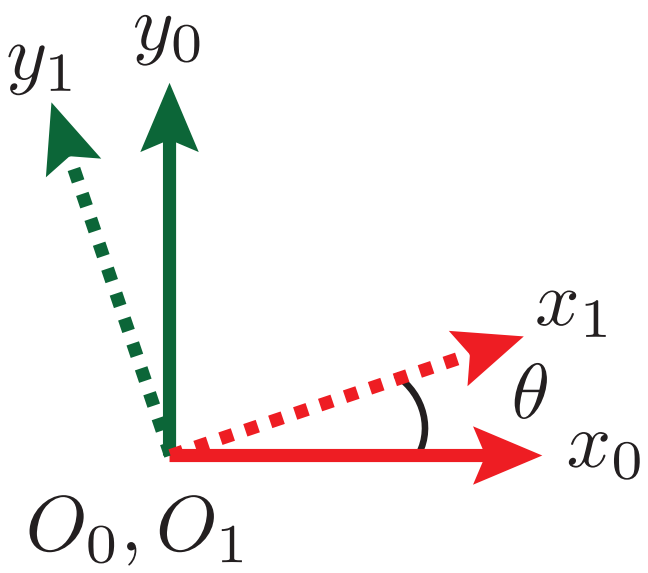
$$p^1 = \begin{bmatrix} 0 \\ 0 \\ -2 \end{bmatrix}$$



$$O_1^0 = \begin{bmatrix} 1 \\ 3 \\ 1 \end{bmatrix}$$

$$O_0^1 = \begin{bmatrix} -1 \\ -1 \\ -3 \end{bmatrix}$$

Planer Rotations

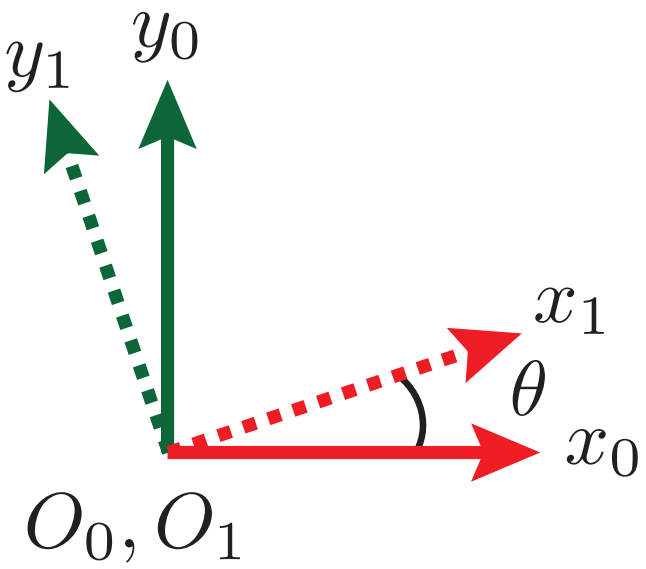


$$R_1^0 = \begin{bmatrix} x_1^0 & | & y_1^0 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

$$x_1^0 = \begin{bmatrix} \cos\theta \\ \sin\theta \end{bmatrix} \qquad y_1^0 = \begin{bmatrix} -\sin\theta \\ \cos\theta \end{bmatrix}$$

Column vectors are unit vectors of frame 1 in frame 0

Planer Rotations



We could also express:

$$x_1^0 = \begin{bmatrix} x_1 \cdot x_0 \\ x_1 \cdot y_0 \end{bmatrix}, y_1^0 = \begin{bmatrix} y_1 \cdot x_0 \\ y_1 \cdot y_0 \end{bmatrix}$$

$$R_1^0 = \begin{bmatrix} x_1 \cdot x_0 & y_1 \cdot x_0 \\ x_1 \cdot y_0 & y_1 \cdot x_0 \end{bmatrix}$$

Because dot product is commutative

$$R_0^1 = (R_1^0)^T$$

Planer Rotations

$$R_0^1 = (R_1^0)^T$$

Geometrically, the orientation of Frame 1 w.r.t Frame 0
Is the inverse of the orientation of Frame 0 w.r.t Frame 1

$$(R_1^0)^T = (R_1^0)^{-1}$$

We refer to this property as orthonormal. Rotation matrices are members of the special orthonormal group $SO(n)$

Properties: $R^T = R^{-1} \in SO(n)$

Mutually orthogonal rows & columns

$\det(R) = 1$

Review 2D Transformations in Homogeneous Coordinates

Rigid Body

Rotation

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Translation

$$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

Scale

$$\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Shear in x

$$\begin{bmatrix} 1 & k_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Shear in y

$$\begin{bmatrix} 1 & 0 & 0 \\ k_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Reflect about y

$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} a & c & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix}$$ General Affine Transformation

More slides on Rotation & Homogeneous Coordinates will be added soon…

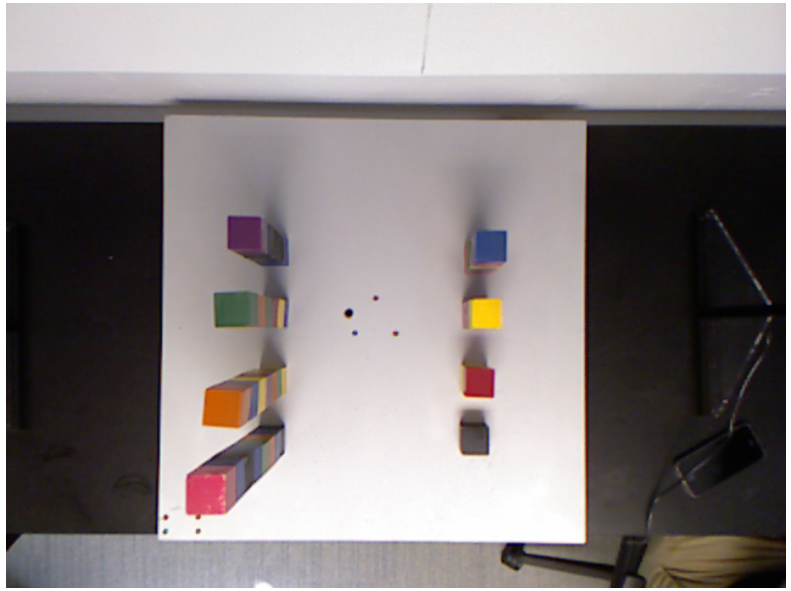
Transforms & Camera Calibration

Lecture #3

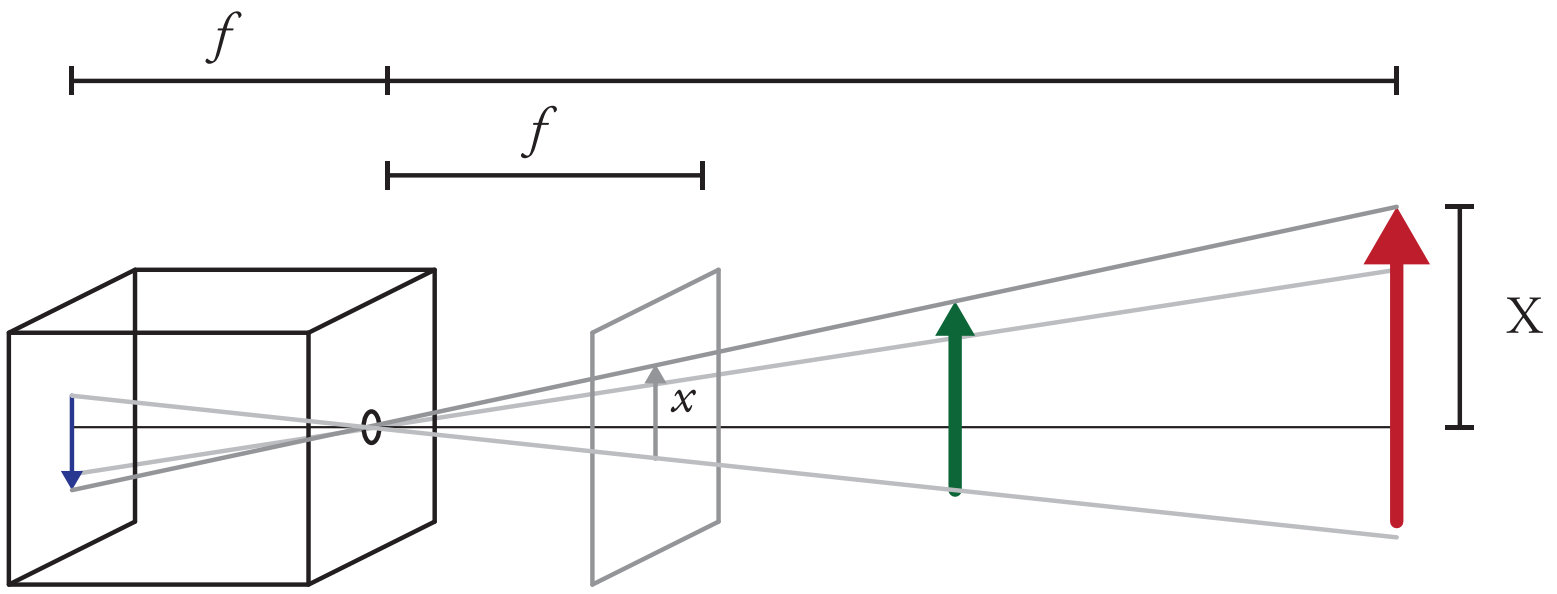
The Problem:

Given an object detected in an image,
what is its location in workspace coordinates?

$$\begin{bmatrix} u \\ v \end{bmatrix} \rightarrow \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

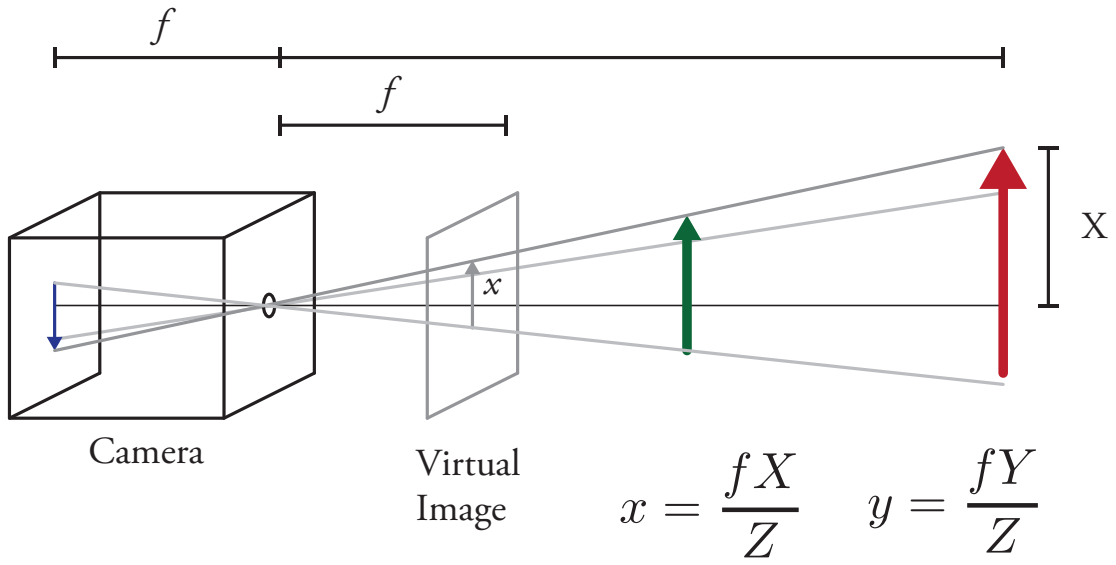


Simple pinhole model of a camera



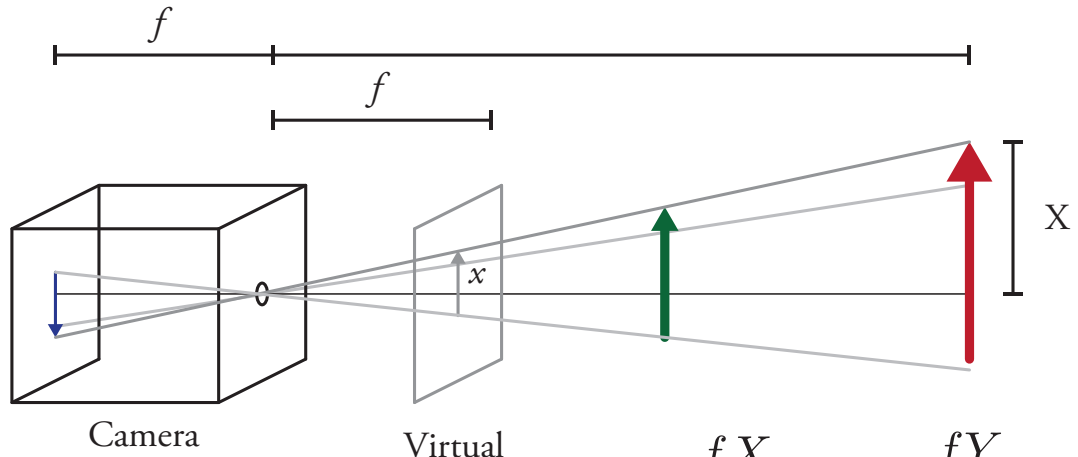
$$x = \frac{fX}{Z} \qquad y = \frac{fY}{Z}$$

In Matrix form



$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \frac{1}{Z} \begin{bmatrix} f & 0 & x_0 & 0 \\ 0 & f & y_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Pinhole Model Intrinsic Matrix

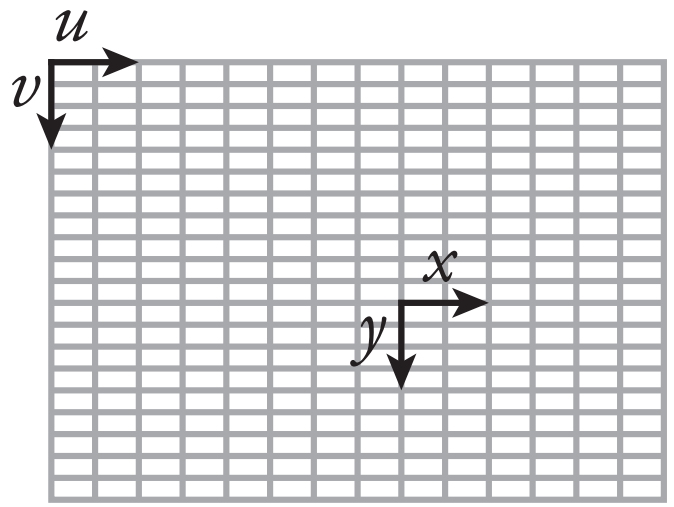


$$x = \frac{fX}{Z} \qquad y = \frac{fY}{Z}$$

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \frac{1}{Z_c} \begin{bmatrix} f & 0 & x_0 \\ 0 & f & y_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{I} & | & 0 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix}$$

Other Considerations…

- Optical axis not centered on sensor
- Pixel coordinate system origin in corner of sensor
- Pixels aren’t square
- Don’t know pixel size



Camera intrinsic matrix

- Transforms Camera Frame to Image Plane
- Depends on geometry of camera, physical characteristics of sensor, details of the image digitization

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \frac{1}{Z_c} \begin{bmatrix} \alpha & s & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{I} & | & 0 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix}$$

Full Projection Matrix

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \mathbf{P} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

intrinsic projection extrinsic

$$\mathbf{P} = \mathbf{K} \times \begin{bmatrix} \mathbf{I} & | & 0 \end{bmatrix} \times \begin{bmatrix} \mathbf{R} & | & \mathbf{t} \end{bmatrix}$$

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \frac{1}{Z_c} \begin{bmatrix} \alpha & s & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{I} & | & 0 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix}$$

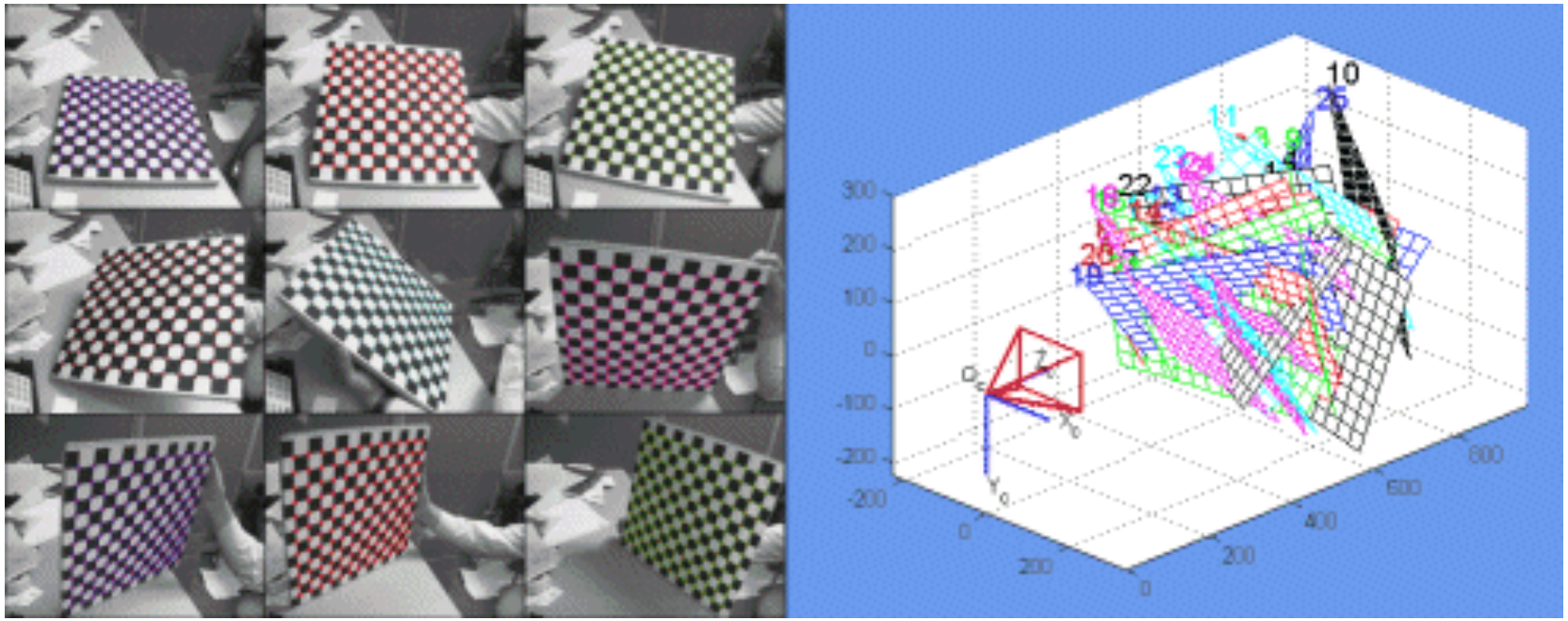
Camera to Image (Intrinsic)

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} = \begin{bmatrix} & \mathbf{R} & & \mathbf{T} \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

World to Camera (Extrinsic)

Finding the intrinsic matrix

- Must perform camera calibration
- Collect many images of known points and compute
- camera_cal.py will give you this (units of pixels)



Jean-Yves Bouguet

Workspace Calibration – Fixed Distance & Camera Aligned

- If the camera Z-axis is aligned with world Z-axis, and all points of interest lie in the X,Y plane a known distance away, we can ignore the Z coordinates
- The Extrinsic matrix can then be expressed as a 2D homogenous transform
- The intrinsic projection matrix then can be expressed as a 3x3 invertible affine transformation

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \frac{1}{Z_c} \begin{bmatrix} f_x & 0 & x_0 & 0 \\ d & f_y & y_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} & \mathbf{R} & & \mathbf{T} \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} X_w \\ Y_w \\ 1 \end{bmatrix}$$

What if we want to go the other way? Pixel -> World

Affine Transformation

- If the camera is directly overhead, and a fixed distance from all objects, we can use an Affine transformation
- Scale, Rotation, Translation, Skew

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

How to find an Affine transform

- Collect a set of known correspondent points – i.e. pixel coordinates of known points on the board

$$(0.3, -0.3) \Leftrightarrow (23, 48)$$

$$\begin{bmatrix} 0.3 \\ -0.3 \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 23 \\ 48 \\ 1 \end{bmatrix}$$

$$(0.0, 0.0) \Leftrightarrow (318, 238)$$

$$\begin{bmatrix} 0.0 \\ 0.0 \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 318 \\ 238 \\ 1 \end{bmatrix}$$

How to find an Affine transform

- Multiply out

$$\begin{bmatrix} 0.3 \\ -0.3 \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 23 \\ 48 \\ 1 \end{bmatrix}$$

$$23a + 48b + c = 0.3$$

$$23d + 48e + f = -0.3$$

$$\begin{bmatrix} 0.0 \\ 0.0 \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 318 \\ 238 \\ 1 \end{bmatrix}$$

$$318a + 238b + c = 0.0$$

$$318d + 238e + f = 0.0$$

How to find an Affine transform

- Rearrange into form $\mathbf{A}x = b$

$$\begin{bmatrix} 23 & 48 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 23 & 48 & 1 \\ 318 & 238 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 318 & 238 & 1 \\ & & \vdots & & & \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \end{bmatrix} = \begin{bmatrix} 0.3 \\ -0.3 \\ 0.0 \\ 0.0 \\ \vdots \end{bmatrix}$$

How to find an Affine transform

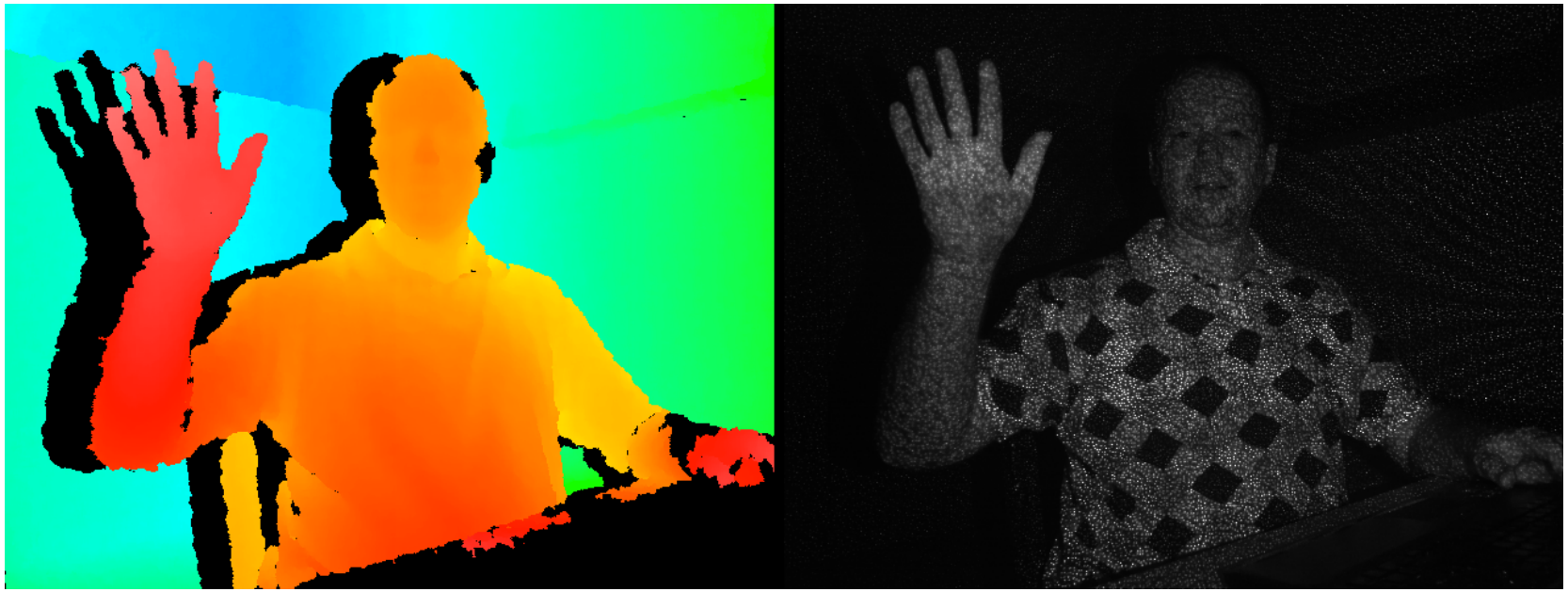
- Solve $\mathbf{A}x = b$

$$\vec{x} = \mathbf{A}^{-1}\vec{b}$$ Invert, Only works if A is square (3 correspondences)

$$\vec{x} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\vec{b}$$ Pseudo inverse, least squares for N>3

Now you have the values to generate the
Affine transformation matrix

Kinect gives depth



- Depth data is a 10 bit number (0-1024)
- Need a function get depth in meters. An example:

$$Z_C = 0.1236 \times \tan(d/2842.5 + 1.1863)$$

Inverse of Intrinsic Matrix

- With depth information we can effectively undo the projection…

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = Z(u,v) \begin{bmatrix} 1/\alpha & 0 & 0 \\ 0 & 1/\beta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -u_0 \\ 0 & 1 & -v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

- This assumes no shear... Otherwise use $\mathbf{K}^{-1}$

Caveat: Depth & RGB camera not aligned

(u, v) pixel in depth image not the same as (u, v) of RGB

Solution: use an affine transform!

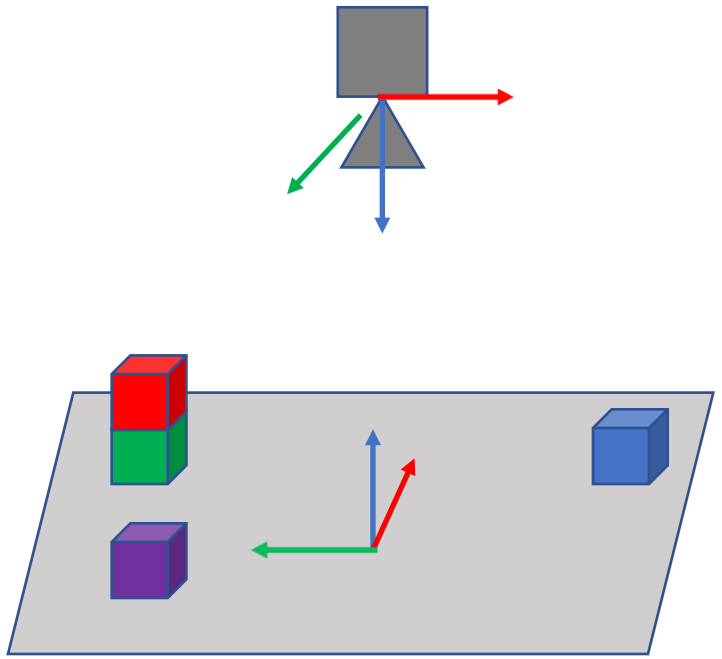
Affine transform to align depth & RGB image

After applying Affine Transform to depth image, RGB pixels and depth pixels are roughly aligned

Can find depth at the centroid of the blocks from the output of your blob detector, look up depth value, and transform (u,v,d) to (x,y,z) using a transformation matrix

Transform Camera Frame to Rexarm Frame

- Use use mouse clicks to align origin and axes to perform this final transformation.

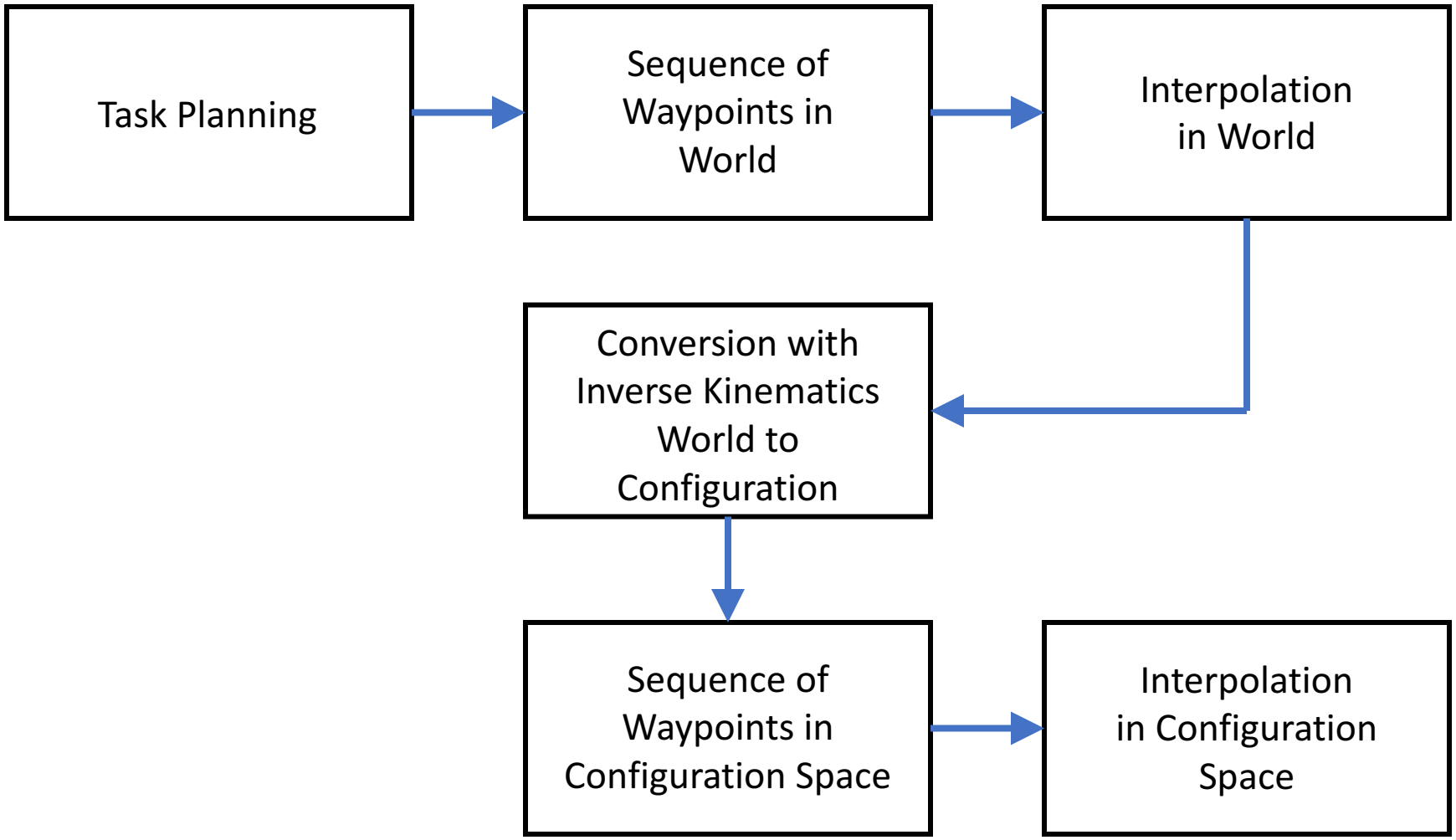


Trajectory Generation

Lecture #4.1

Spong 5.5

Task to Trajectory



World Coordinates

Conversion with Inverse Kinematics

Configuration Space

Trajectory Smoothing

- Smoothing trajectories allows:
  - Limiting accelerations
  - Limiting vibrations
  - Higher precision
  - Less wear in motors

Cubic Polynomial Trajectories

- Given boundary conditions define a polynomial with 4 coefficients:

$$\text{Constraints} : q_0, q_f, v_0, v_f$$

$$q(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3$$

$$\dot{q}(t) = a_1 + 2a_2 t + 3a_3 t^2$$

Cubic Polynomial Trajectories

- Plugging in t_0 and t_f this gives us 4 equations w/ 4 unknowns

$$q_0 = a_0 + a_1 t_0 + a_2 t_0^2 + a_3 t_0^3$$
$$v_0 = a_1 + 2a_2 t_0 + 3a_3 t_0^2$$
$$q_f = a_0 + a_1 t_f + a_2 t_f^2 + a_3 t_f^3$$
$$v_f = a_1 + 2a_2 t_f + 3a_3 t_f^2$$

Cubic Polynomial Trajectories

- Can be rearranged in matrix form to solve for *a*

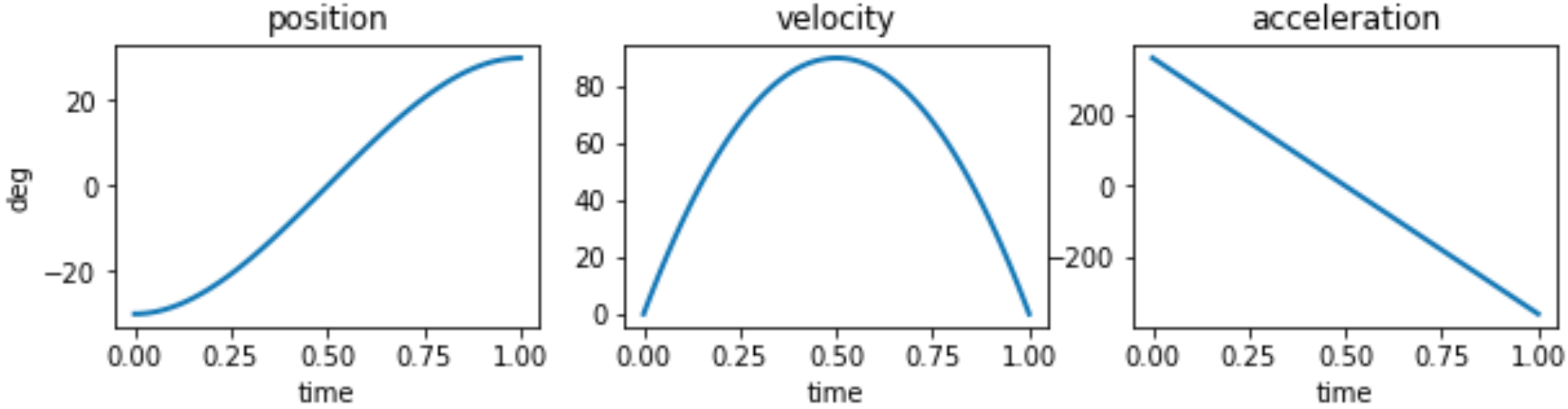
$$\begin{bmatrix} 1 & t_0 & t_o^2 & t_0^3 \\ 0 & 1 & 2t_0 & 3t_0^2 \\ 1 & t_f & t_f^2 & t_f^3 \\ 0 & 1 & 2t_f & 3t_f^2 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} q_0 \\ v_0 \\ q_f \\ v_f \end{bmatrix}$$

$$\mathbf{M}a = b$$

$$a = \mathbf{M}^{-1}b$$

Cubic Polynomial Trajectories

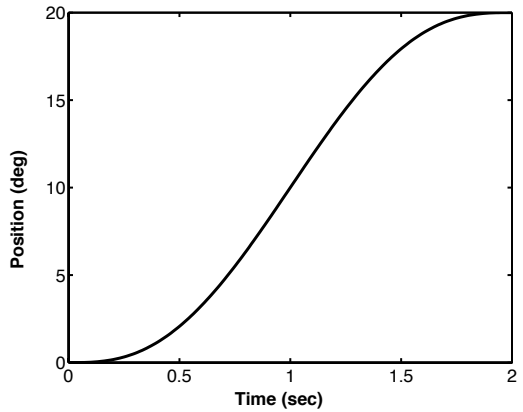
- Cubic polynomial creates smooth position and velocity curves
- Acceleration is discontinuous
- System naturally limits acceleration (due to inertia)
- Still large jerk (derivative of acceleration) can be harmful

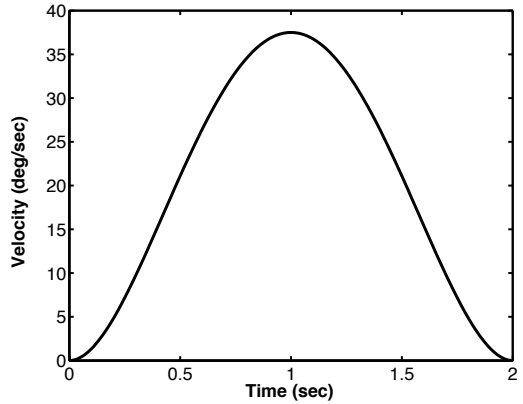


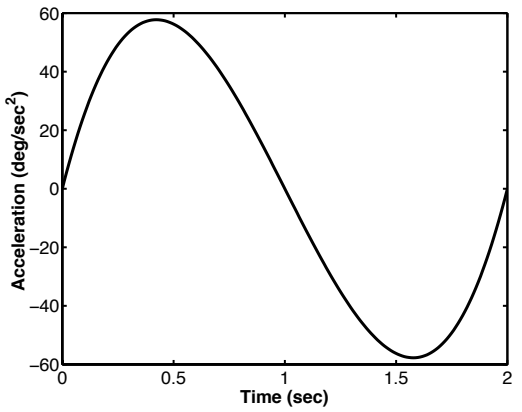
Quintic Polynomial

- Allows boundary condition on acceleration
- Limits Jerk

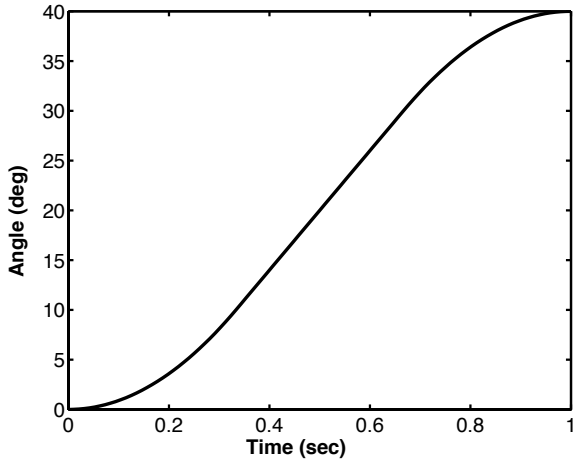
$$\begin{bmatrix} 1 & t_0 & t_0^2 & t_0^3 & t_0^4 & t_0^5 \\ 0 & 1 & 2t_0 & 3t_0^2 & 4t_0^3 & 5t_0^4 \\ 0 & 0 & 2 & 6t_0 & 12t_0^2 & 20t_0^3 \\ 1 & t_f & t_f^2 & t_f^3 & t_f^4 & t_f^5 \\ 0 & 1 & 2t_f & 3t_f^2 & 4t_f^3 & 5t_f^4 \\ 0 & 0 & 2 & 6t_f & 12t_f^2 & 20t_f^3 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{bmatrix} = \begin{bmatrix} q_0 \\ v_0 \\ \alpha_0 \\ q_f \\ v_f \\ \alpha_f \end{bmatrix}$$

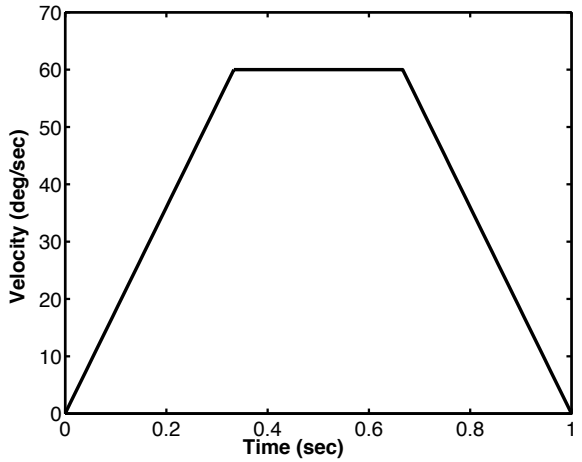


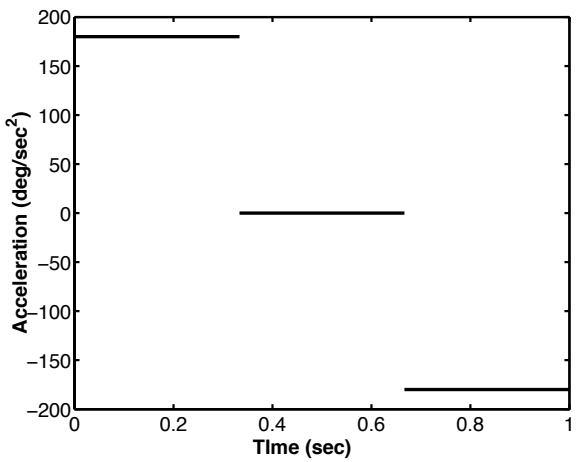




Linear Segments with Parabolic Blends

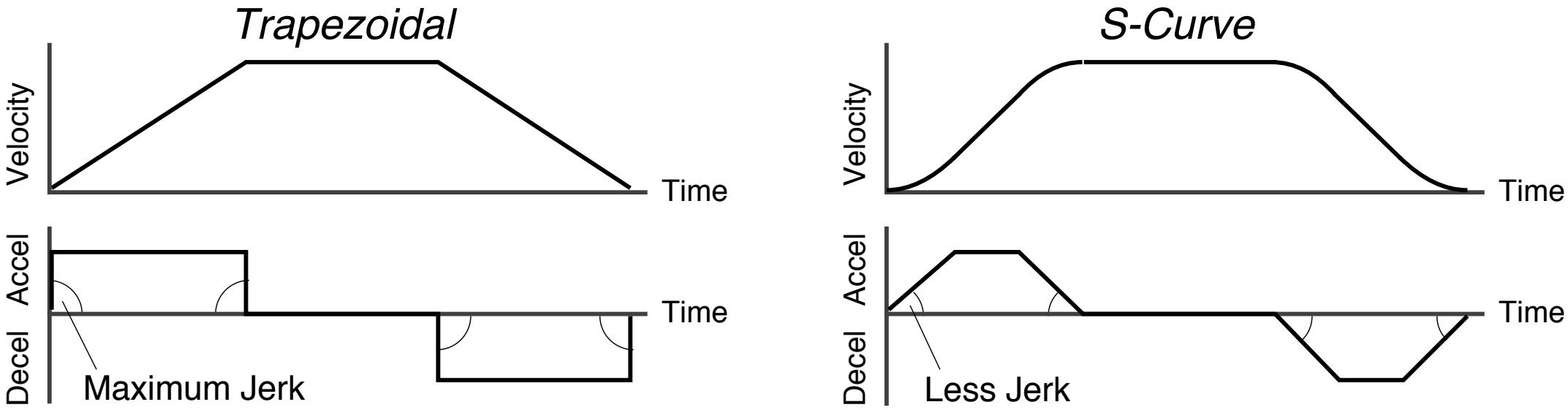






$$q(t) = \begin{cases} q_0 + \frac{a}{2}t^2 & 0 \le t \le t_b \\ \frac{q_f + q_0 - Vt_f}{2} + Vt & t_b < t \le t_f - t_b \\ q_f - \frac{at_f^2}{2} + at_f t - \frac{a}{2}t^2 & t_f - t_b < t \le t_f \end{cases}$$

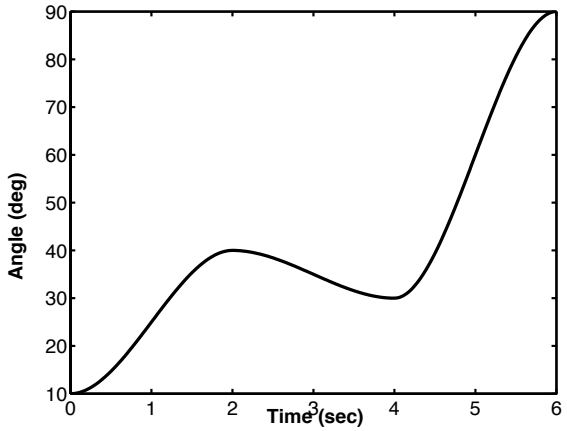
Trapezoidal vs. S-Curve

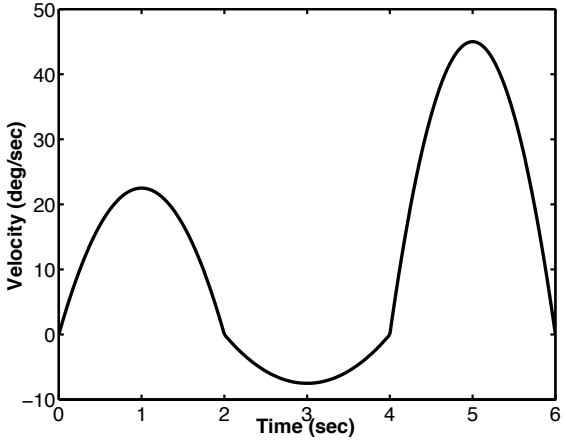


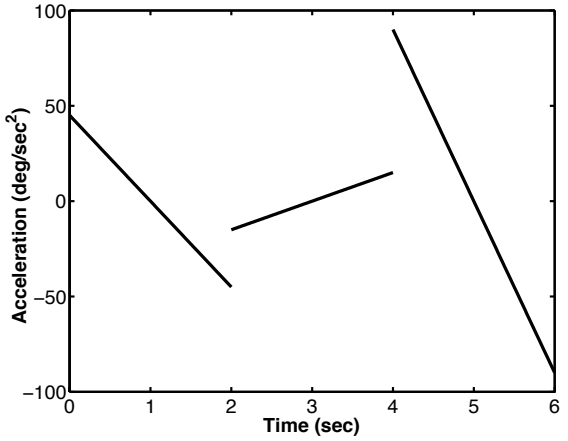
Using parabolic blends on the trapezoidal velocity profile gives an S-Curve profile with a trapezoidal acceleration profile. This limits jerk.

Stitching together waypoints

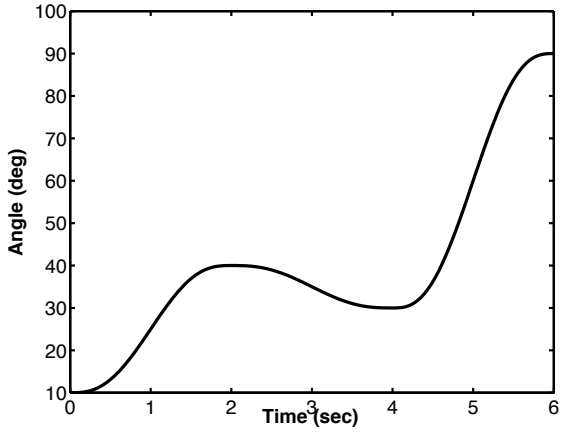
Cubic Spline

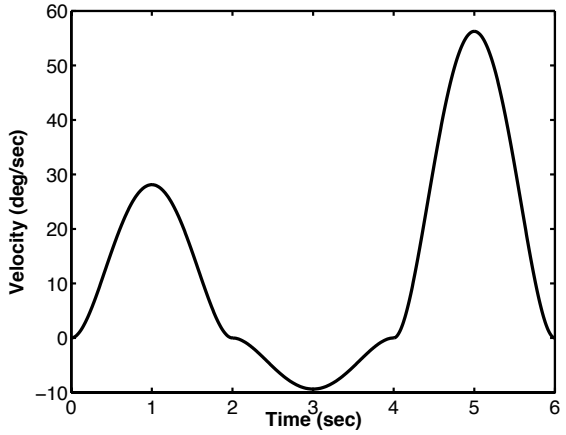


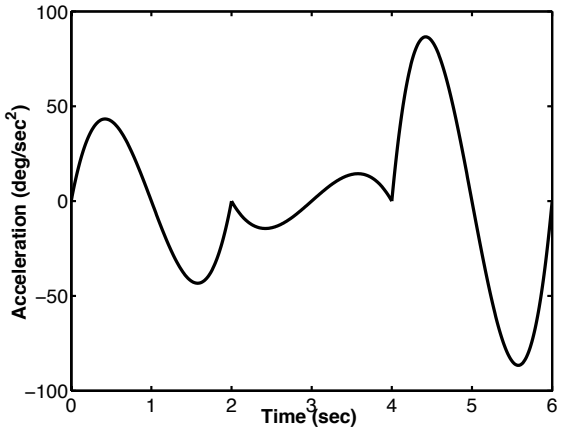




Quintic Spline

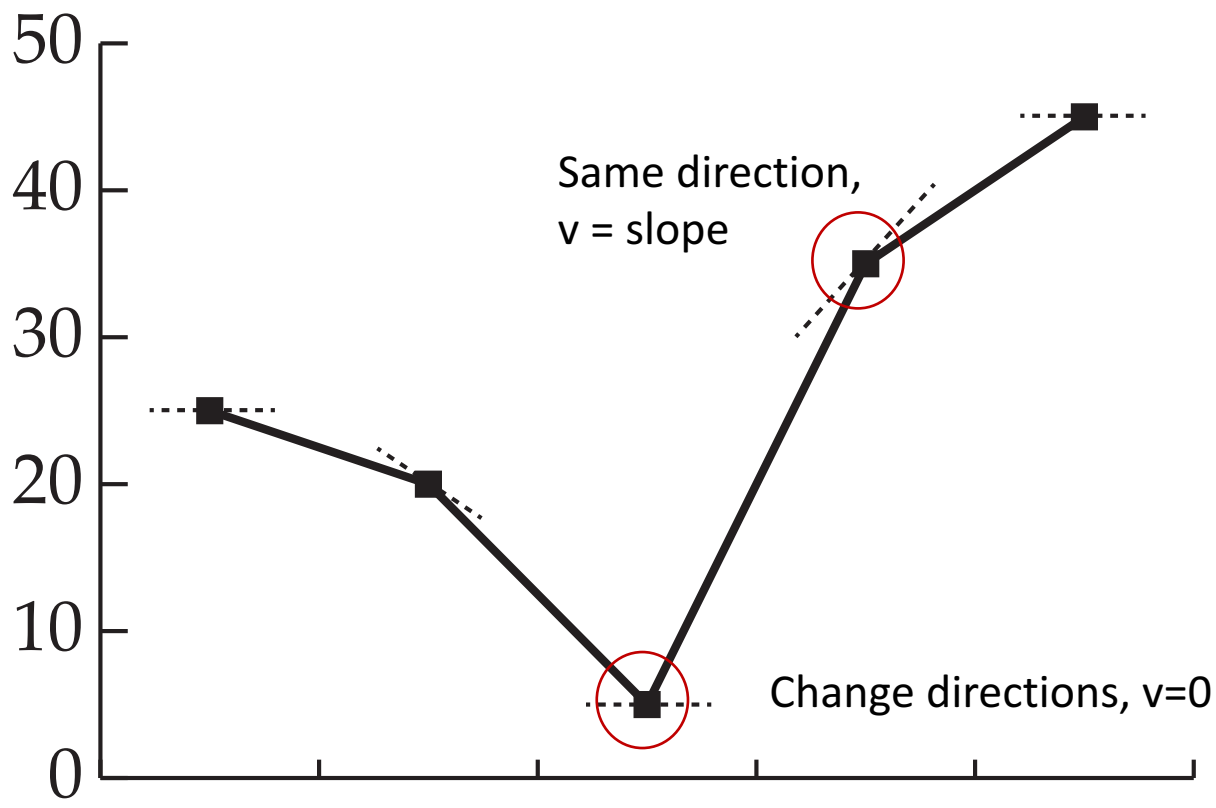




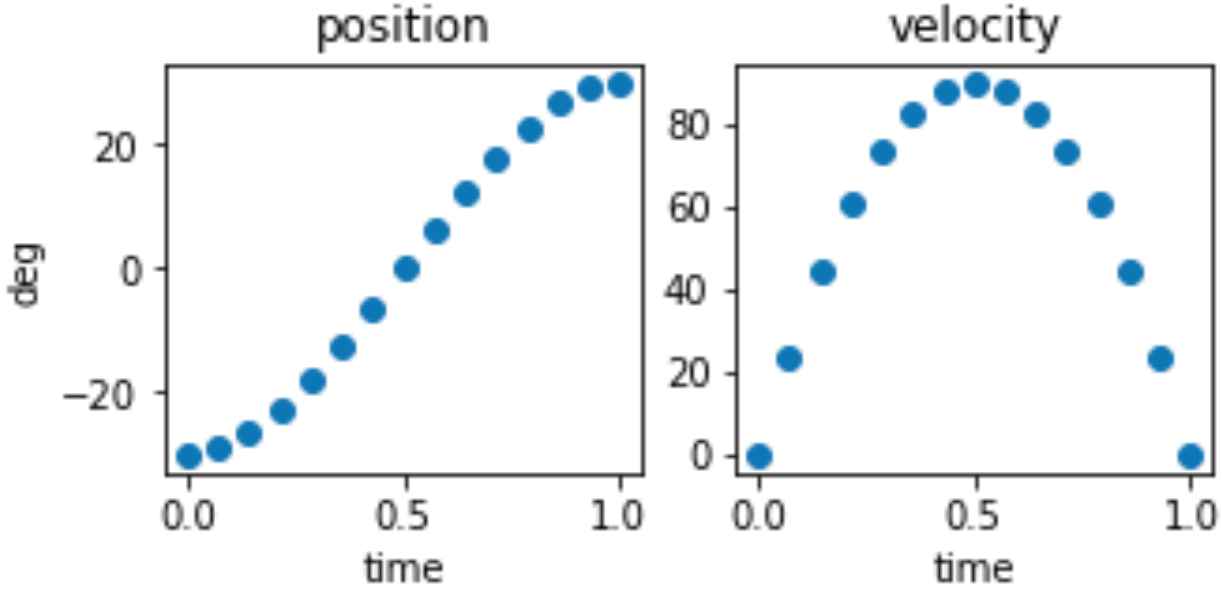


Ways to choose velocity at waypoints

- User specifies velocities in terms of angular velocities for each motor (could be 0)
- System chooses velocities at the waypoints to ensure acceleration through the points is continuous



Implementation



- “Look ahead” when commanding position waypoints and set the speed command to the desired velocity at each time step.
- When deciding time, look at largest joint displacement and and decide some metric…
- When reporting “low” and ”high” speed playback, you should be adjusting this metric

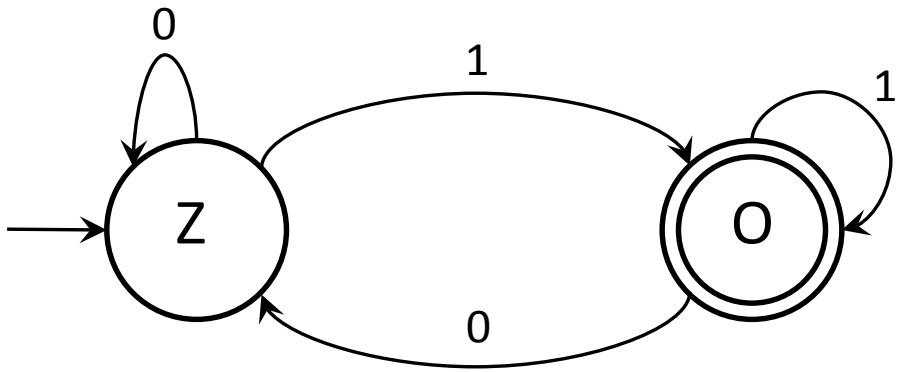
Deterministic Finite State Machines

Lecture #4.2

Deterministic Finite State Automaton

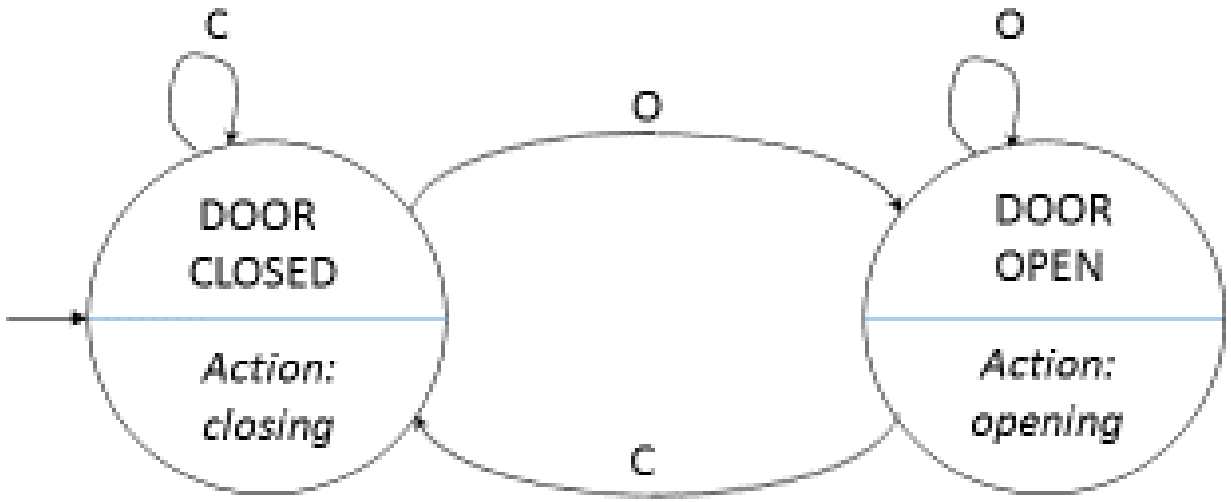
The deterministic finite state automaton (DFSA) is defined as the tuple D = (Q, Σ, δ, q_0, F) where:

- Q is the finite non-empty set of states (graph nodes)
- Σ is the finite non-empty alphabet
- $q_0 \in Q$ is the unique initial state
- $F \subseteq Q$ is the finite and possibly empty set of final states[1]
- $\delta = Q \times \Sigma \rightarrow Q$ is the total transition function mapping single input characters and the 'current' state to a new state. The transition function δ defines the DFSA graph edges.

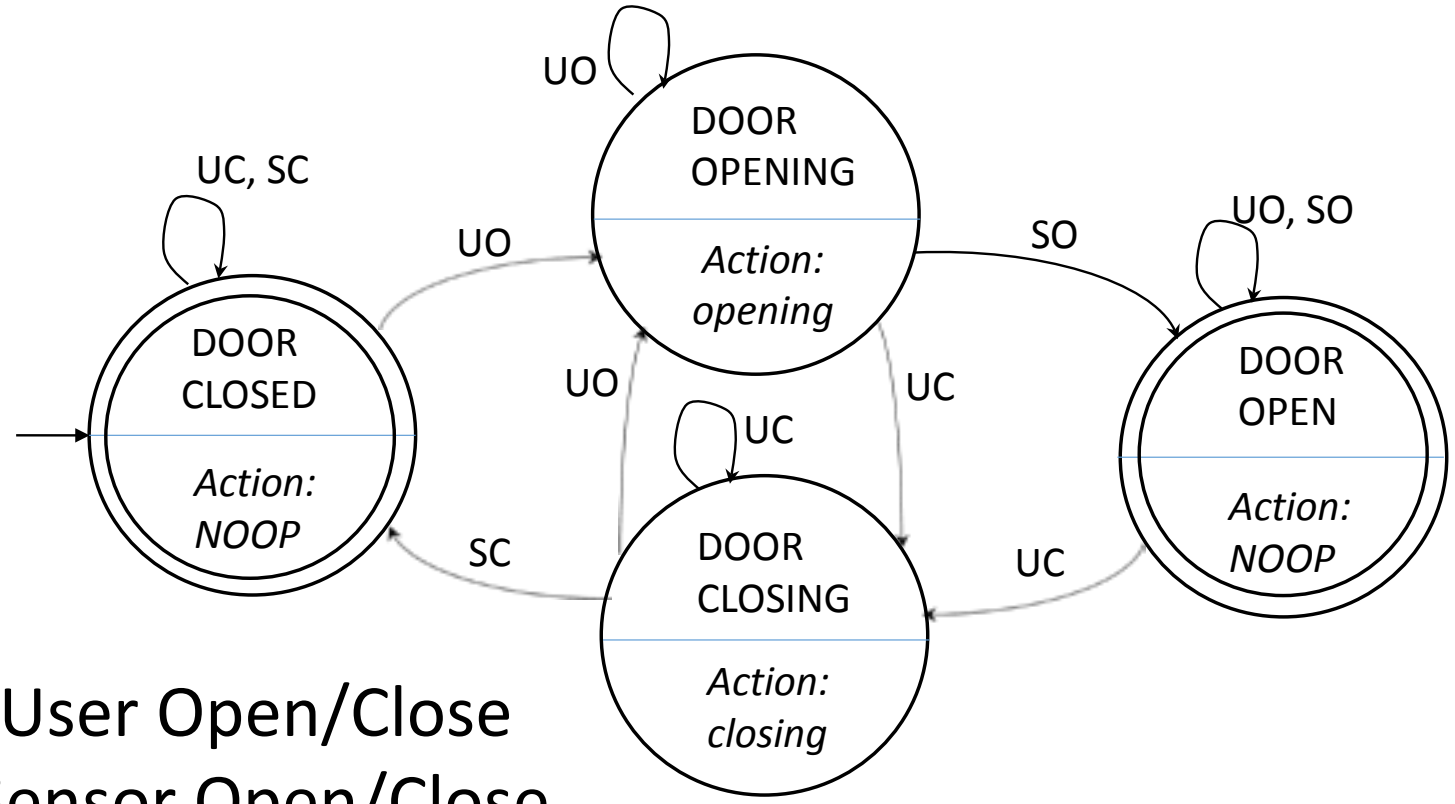


$$Q = \{Z, O\}$$
$$\Sigma = \{0, 1\}$$
$$q_0 = Z$$
$$\delta = \{< Z, 0, Z >, < Z, 1, O >, < O, 0, Z >, < O, 1, O >\}$$

Simple Example – Elevator Door



Adding Intermediate States



UO / UC = User Open/Close
SO / SC = Sensor Open/Close

Implementation in Switch Case

```
switch(currentState) {
    case DOOR_CLOSING:
        close_door();
        if(sensor = closed)
            currentState = DOOR_CLOSED;
        if(button = open)
            currentState = DOOR_OPENING;
    case DOOR_OPENING:
        open_door()
        if(sensor = open)
             currentState = DOOR_OPEN;
        if(button = close)
            currentState = DOOR_CLOSING;
    case DOOR_CLOSED:
        if(button = open)
            currentState = DOOR_OPENING;
    case DOOR_OPEN:
        if(button = close)
            currentState = DOOR_CLOSING;
    default:
        throw_error();
```

Let’s Draw a State Machine for Moving the Arm!

Forward Kinematics

Lecture #5.1

Spong CH3.1 – 3.2

Review Rigid Body Homogeneous Transformations

$$\mathbf{H} = \begin{bmatrix} & \mathbf{R} & & \mathbf{d} \\ 0 & 0 & 0 & 1 \end{bmatrix}; \mathbf{R} \in SO(3), d \in \mathbb{R}^3$$

$$\mathbf{H}^{-1} = \begin{bmatrix} & \mathbf{R}^T & & -\mathbf{R}^T\mathbf{d} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Building the **R** matrix

- Basic Rotation Matrices

$$\mathbf{R}_{x,\psi} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c_\psi & -s_\psi \\ 0 & s_\psi & c_\psi \end{bmatrix}$$

$$\mathbf{R}_{y,\theta} = \begin{bmatrix} c_\theta & 0 & s_\theta \\ 0 & 1 & 0 \\ -s_\theta & 0 & c_\theta \end{bmatrix}$$

$$\mathbf{R}_{z,\phi} = \begin{bmatrix} c_\phi & -s_\phi & 0 \\ s_\phi & c_\phi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Denavit – Hartenberg Convention

$$\mathbf{A}_i^{i-1}(q_i) = \begin{bmatrix} c_{\theta_i} & -s_{\theta_i}c_{\alpha_i} & s_{\theta_i}s_{\alpha_i} & a_i c_{\theta_i} \\ s_{\theta_i} & c_{\theta_i}c_{\alpha_i} & -c_{\theta_i}s_{\alpha_i} & a_i s_{\theta_i} \\ 0 & s_{\alpha_i} & c_{\alpha_i} & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

More slides on Kinematics will be added soon…

Inverse Kinematics

Lecture #5.2

Spong CH3.3

More slides on Kinematics will be added soon…

Differential Kinematics

Lecture #6.1

Spong CH4

Differential Kinematics

- Motivation: consider an n-link serial chain manipulator with joint variables:

$$(q_1, q_2, \ldots, q_n)$$

- This manipulator can be described by the forward kinematic matrix

$$\mathbf{T}_n^0(\mathbf{q}) = \begin{bmatrix} R_n^0(q) & o_n^0(q) \\ 0 & 1 \end{bmatrix}$$

End Effector Velocity

- Want to find the end effector velocity given the joint velocities

$$\xi = \begin{bmatrix} v_n^0 \\ \omega_n^0 \end{bmatrix} = \begin{bmatrix} v_x \\ v_y \\ v_z \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} \qquad \text{given} \quad \dot{q} = \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \vdots \\ \dot{q}_n \end{bmatrix}$$

The Jacobian

- Forward kinematics gives us

$$\mathbf{T}_n^0(\mathbf{q}) = A_1(q_1)A_2(q_2)...A_n(q_n)$$

- If we want to know how moving one joint moves our end effector…

$$\frac{\partial \mathbf{T}_n^0(\mathbf{q})}{\partial q_k} = A_1(q_1)...\frac{\partial A_k}{\partial q_k}A_k(q_k)...A_n(q_n)$$

- The expression for this for all joint variables is the Jacobian

Building the Geometric Jacobian

- The linear velocity of the end effector frame is

$$v_n^0 = J_v \dot{q}$$

Is 3 x n matrix

- The angular velocity of the end effector frame is

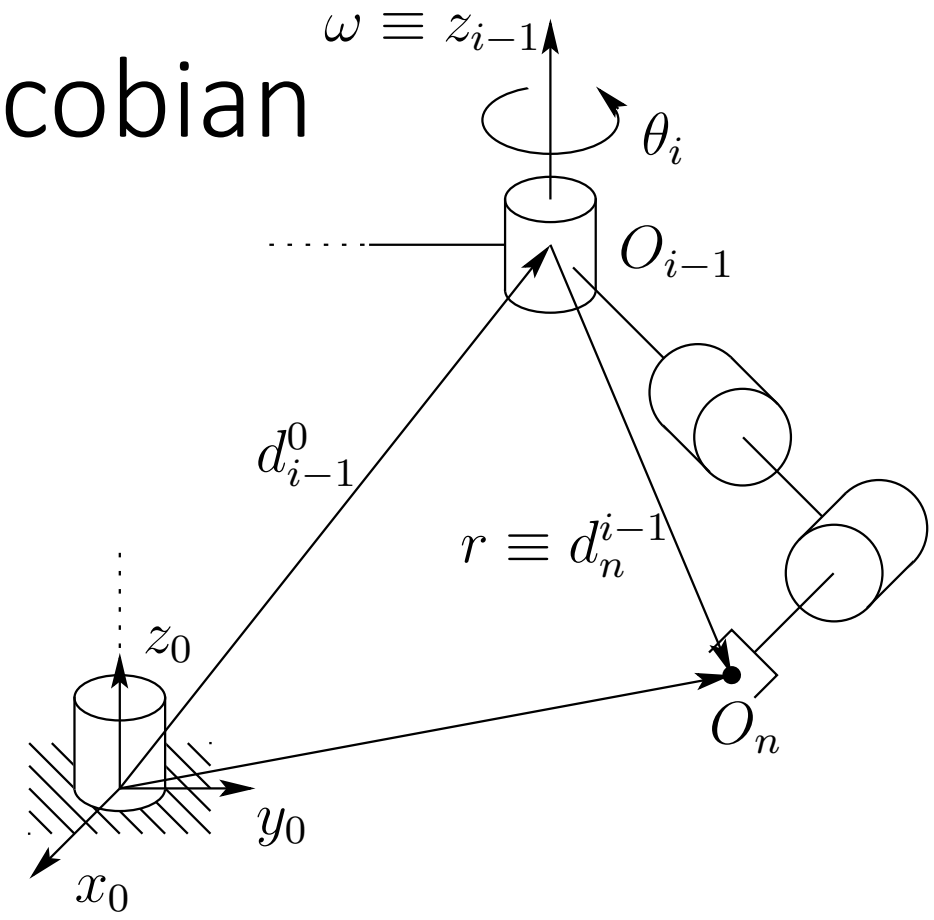
$$\omega_n^0 = J_\omega \dot{q}$$

is 3 x n matrix

$$J = \begin{bmatrix} J_v \\ J_\omega \end{bmatrix}$$

is 6 x n matrix

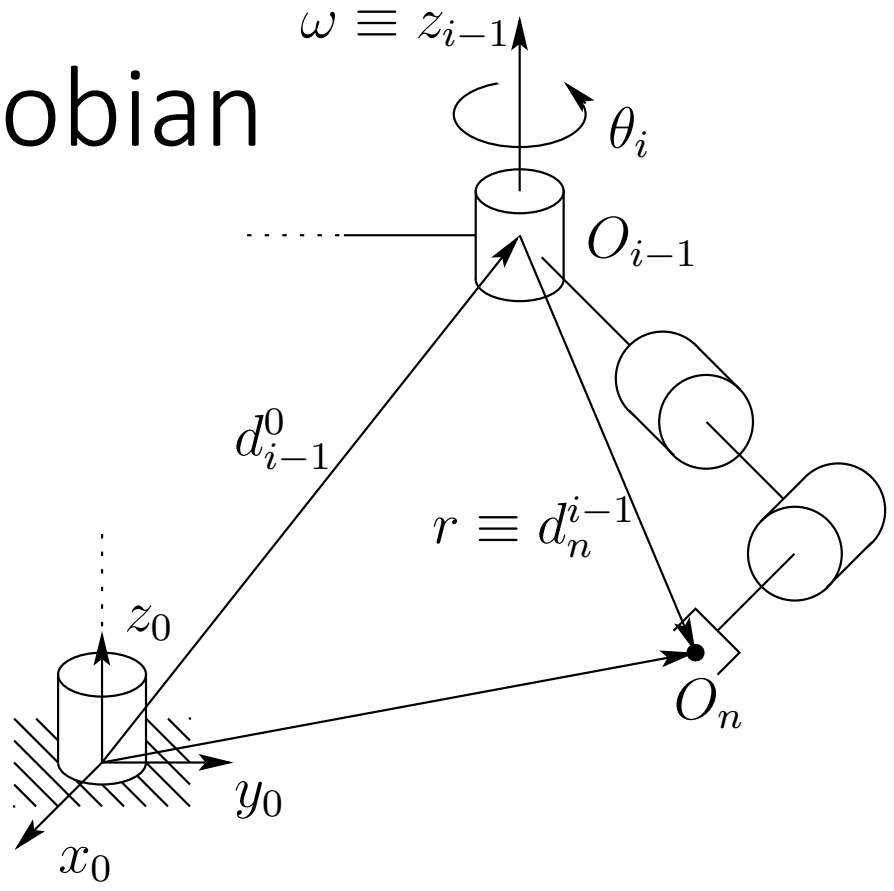
Building the Geometric Jacobian



the i-th column J_{ω_i} is

$$J_{\omega_i} = \begin{cases} z_{i-1} & \text{for revolute joint } i \\ 0 & \text{for prismatic joint } i \end{cases}$$

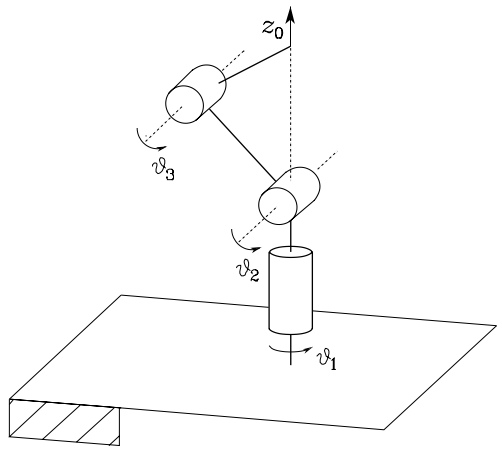
Building the Geometric Jacobian



the i-th column J_{v_i} is

$$J_{v_i} = \begin{cases} z_{i-1} \times (o_n - o_{i-1}) & \text{for revolute joint } i \\ z_{i-1} & \text{for prismatic joint } i \end{cases}$$

Anthropomorphic Arm



- 3DOF Serial revolute manipulator

$$J = \begin{bmatrix} z_0 \times (o_3 - o_0) & z_1 \times (o_3 - o_1) & z_2 \times (o_3 - o_2) \\ z_0 & z_1 & z_2 \end{bmatrix}$$

$$\boldsymbol{J} = \begin{bmatrix} -s_1(a_2c_2 + a_3c_{23}) & -c_1(a_2s_2 + a_3s_{23}) & -a_3c_1s_{23} \\ c_1(a_2c_2 + a_3c_{23}) & -s_1(a_2s_2 + a_3s_{23}) & -a_3s_1s_{23} \\ 0 & a_2c_2 + a_3c_{23} & a_3c_{23} \\ 0 & s_1 & s_1 \\ 0 & -c_1 & -c_1 \\ 1 & 0 & 0 \end{bmatrix}$$

Singularities

- The 6 x n Jacobian defines a mapping

$$\xi = J(q)\dot{q}$$

- This implies all possible end-effector velocities are linear combinations of the columns of the Jacobian

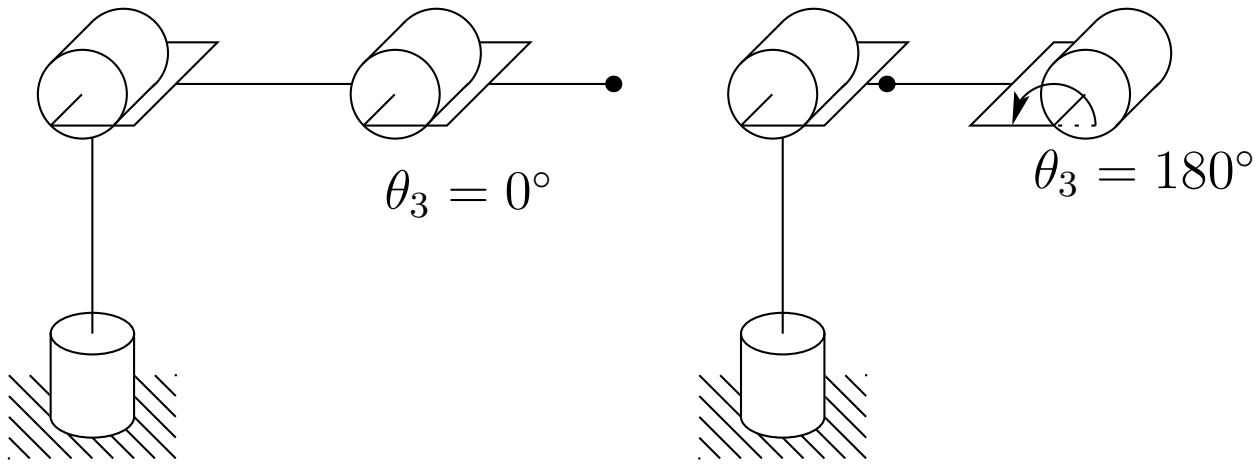
$$\xi = J_1\dot{q_1} + J_2\dot{q_2} + ... + J_n\dot{q_n}$$

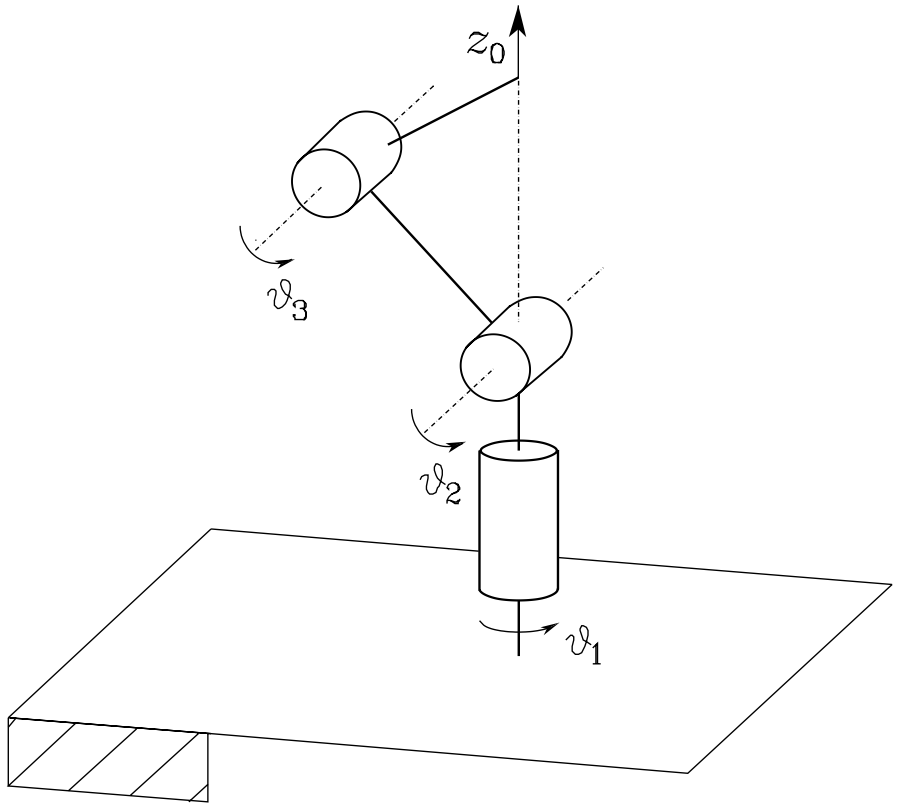
- When J has rank 6, we can move with arbitrary velocities, but when rank(J) < 6 or <n for an n-DOF arm, we are in a “singular” configuration

Singularities

- Singularities represent configurations where certain directions of motion are unobtainable
- At singularities, bounded end-effector velocities may correspond to unbounded joint velocities
- At singularities, bounded end-effector torques may correspond to unbounded joint torques.
- Singularities usually correspond to points on the boundary of the workspace.
- Singularities may correspond to points in the workspace that are unreachable.
- Near a singularity there will not exist a unique solution to the inverse kinematics problem. There may be no solution or infinite solutions.

Singularities





Inverse Velocity and Acceleration

- The Jacobian relates the joint velocity to the end effector velocity

$$\xi = J\dot{q}$$

- For a square Jacobian we can invert it and find joint velocities given a velocity we with the end-effector to have

$$\dot{q} = J^{-1}\xi$$

- Even if it is not square, it turns out you can use the pseudo inverse

$$J^{+} = J^{T}(JJ^{T})^{-1}$$

$$\dot{q} = J^{+}\xi$$

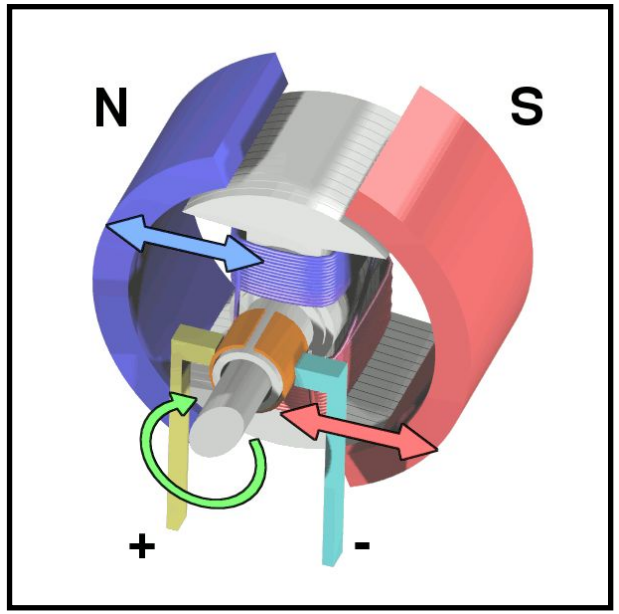
Differential Inverse Kinematics

- This relationship allows differential inverse kinematics
- Paths in workspace can be achieved by following an end-effector velocity profile when joint velocity commands are found using the inverse Jacobian
- This scheme works as long as singularities are avoided
- Often this is the simplest way to plan complex paths in workspace, especially for kinematically redundant manipulators.

Balancebot Hardware Intro

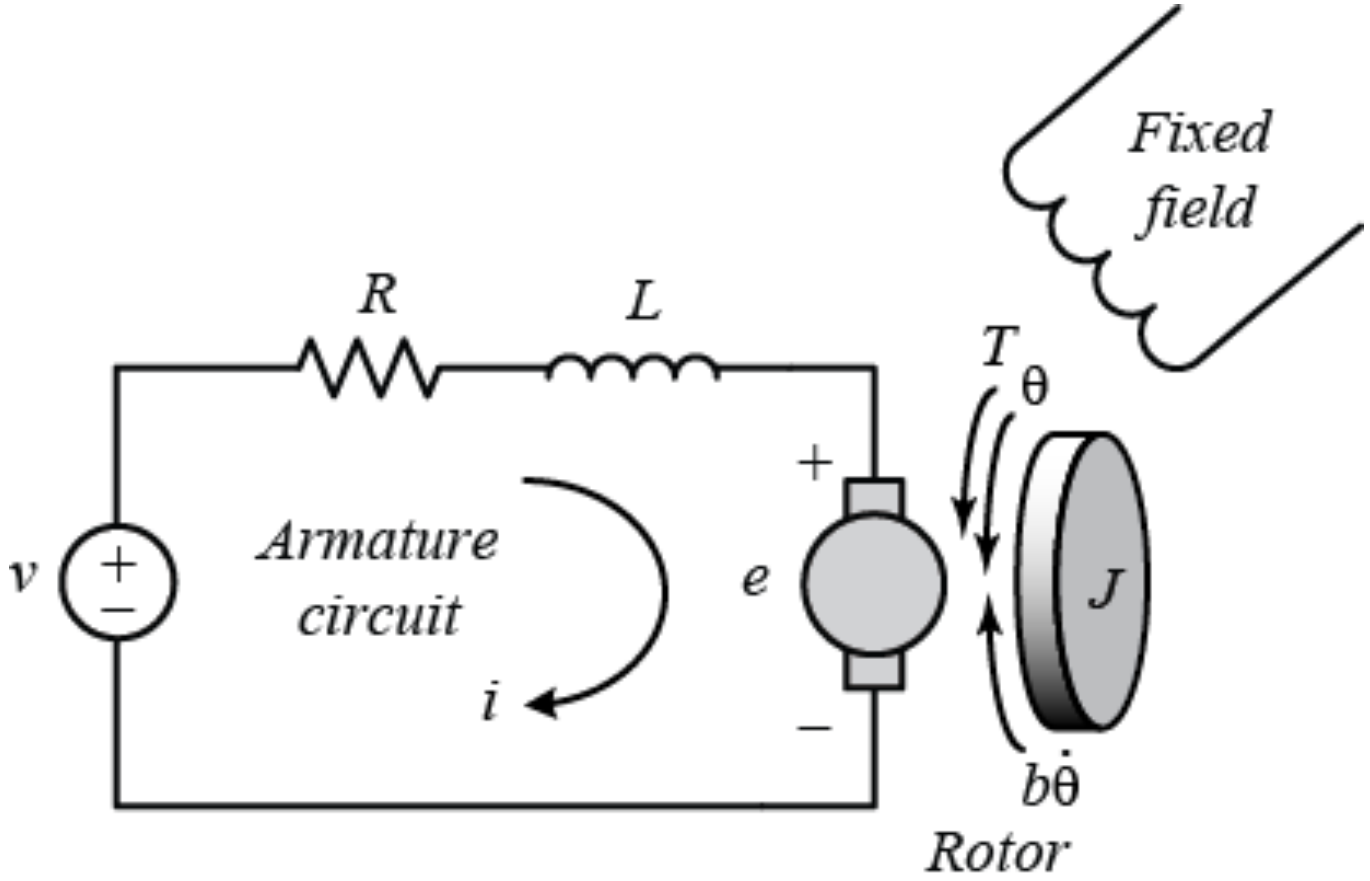
Brushed DC Motor - Basics

- DC current produces magnetic fields in armature
- Magnetic fields align with those of the permanent magnets
- Commutator switches direction of current, and thus polarity of magnetic field to keep motor rotating
- Contact to the commutator made with metal brushes
- As motor spins, field from permanent magnets creates a back EMF in the coils (generator)



Motor Model

- R - armature resistance (Ω)
- L - armature inductance (H)
- v – applied voltage (V)
- e – back EMF (V)
- T – motor torque (N.m)
- θ - position (rad)
- b – friction coefficient (N.m)



Motor Model

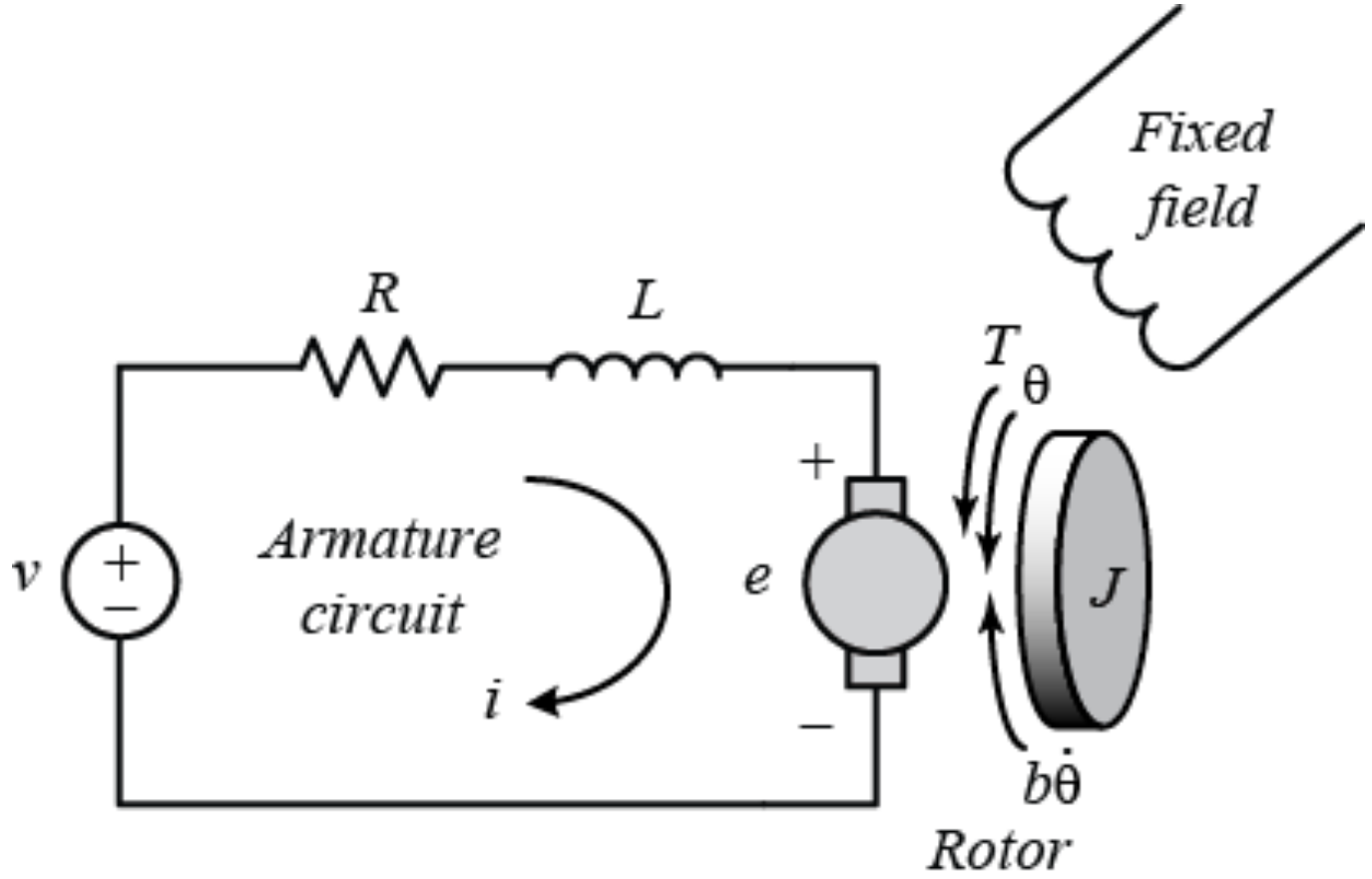
$$T = K_t i$$

$$e = K_e \dot{\theta}$$

$K_t = K_e$ in S.I. units

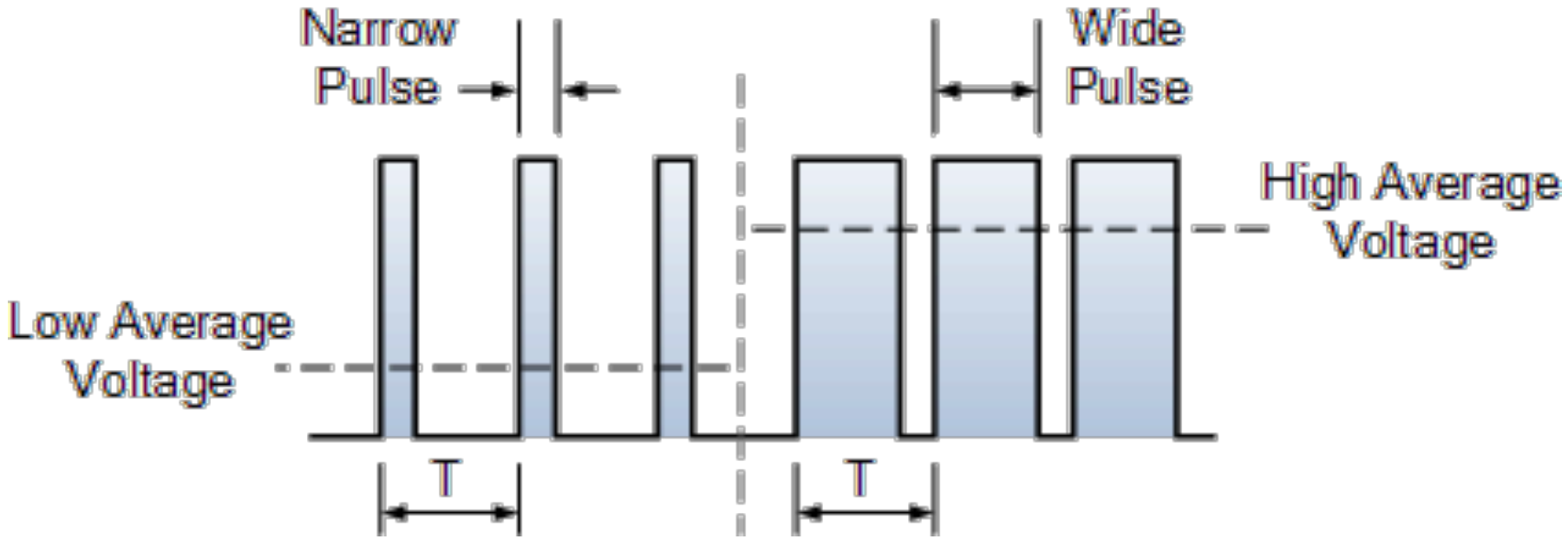
$$Ki = J\ddot{\theta} + b\dot{\theta}$$

$$L\frac{di}{dt} + Ri = V - K\dot{\theta} \qquad V = L\frac{di}{dt} + Ri + K\dot{\theta}$$

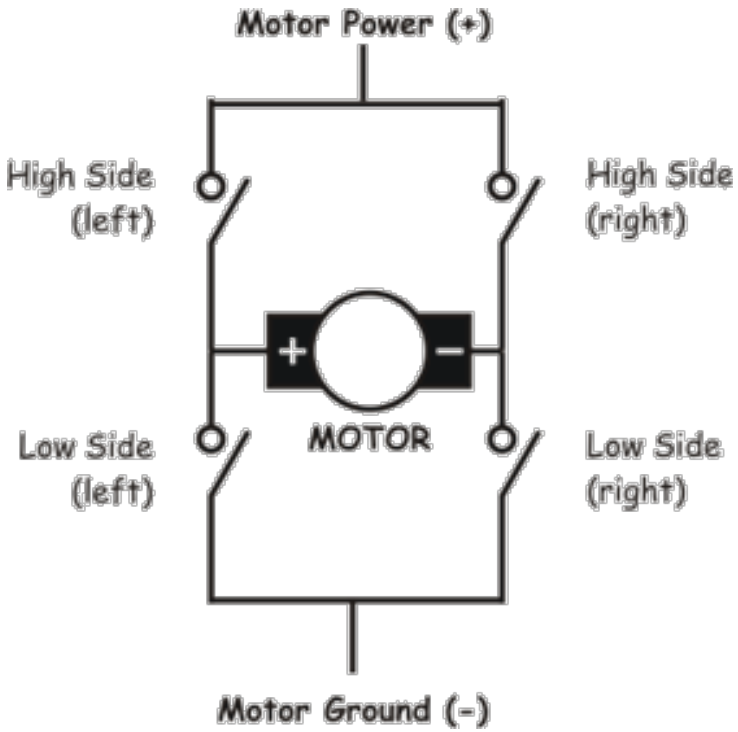


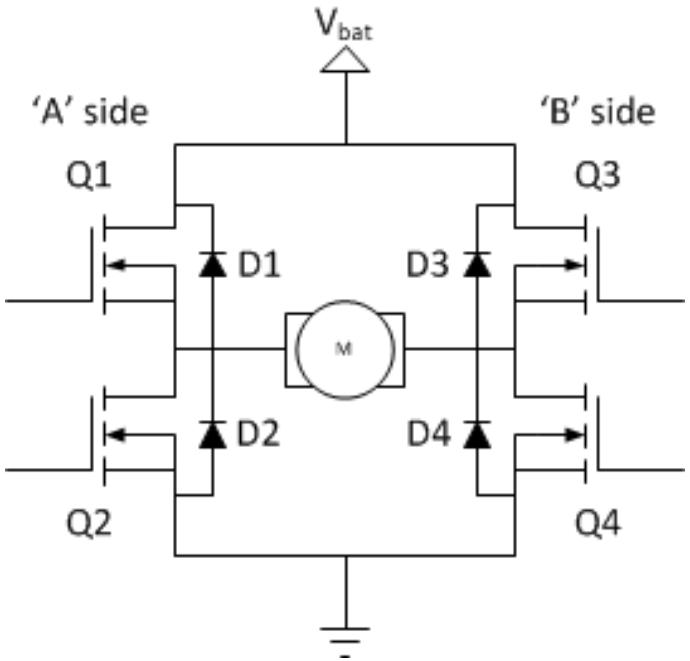
Pulse Width Modulation (PWM)

- Motor acts as a low pass filter (R-L filter)
- Can control speed with a “digital” PWM signal at high frequency
- Lower inductance motors require higher frequency
- PWM has a frequency/period & a duty cycle/pulse width

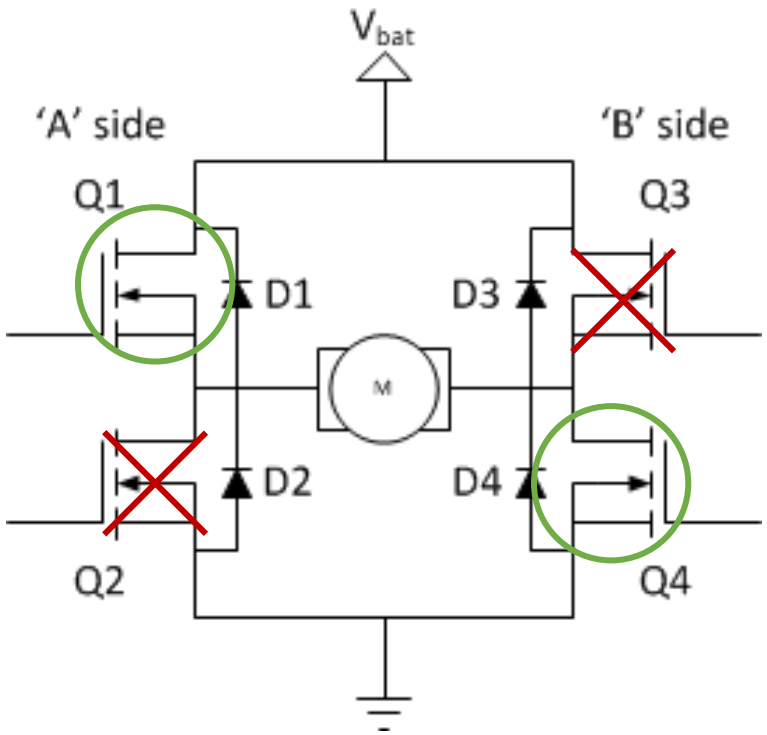


H-Bridge

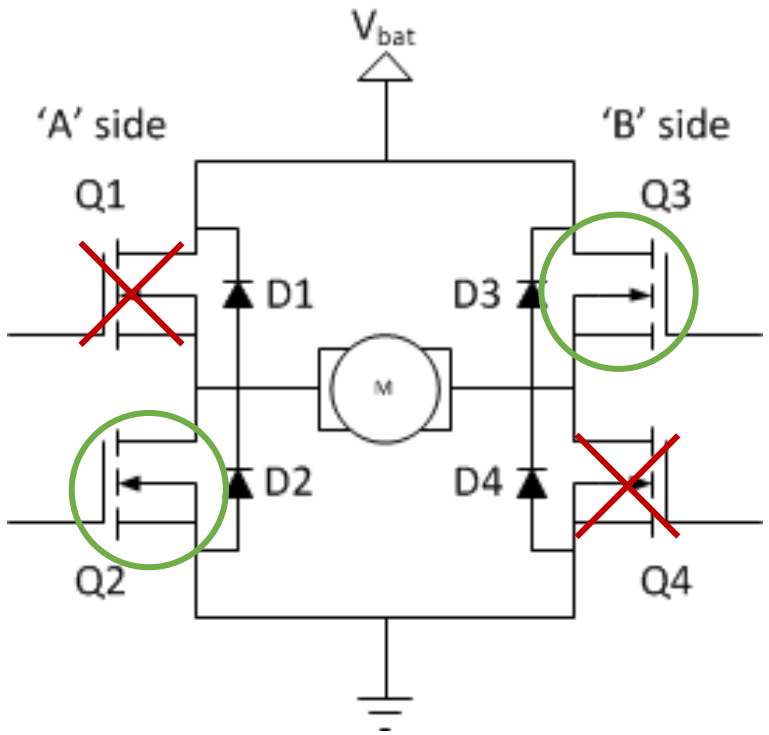

Motor Power (+)
High Side (left)
High Side (right)
MOTOR
Low Side (left)
Low Side (right)
Motor Ground (-)


V_{bat}
'A' side
'B' side
Q1
Q3
D1
D3
M
D2
D4
Q2
Q4

Normal Operation

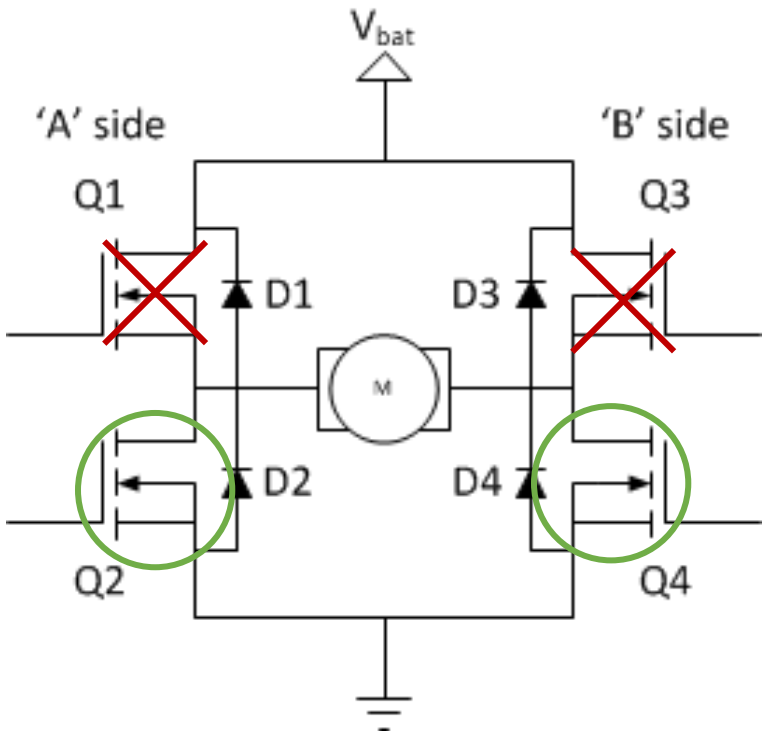


Forward

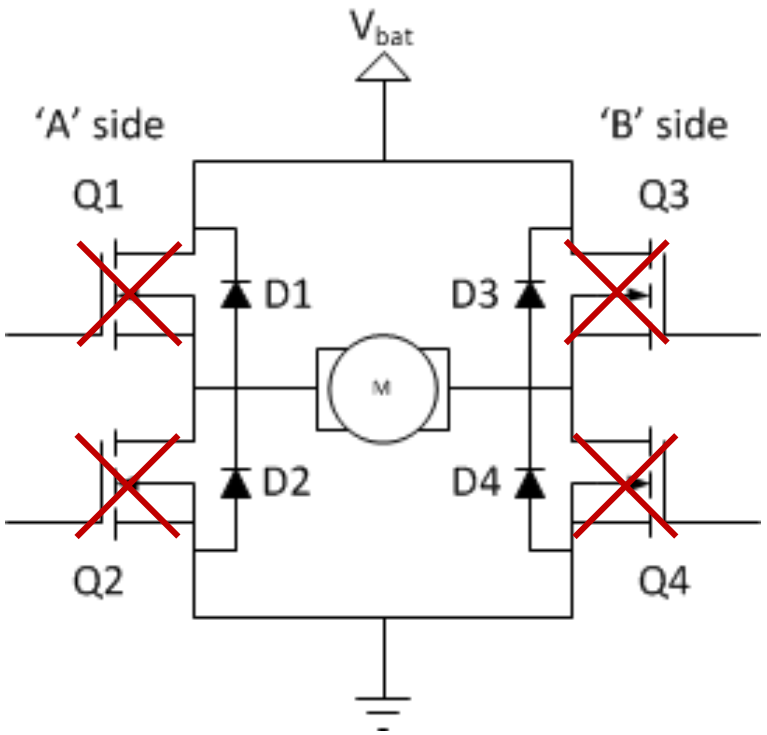


Reverse

Normal Operation

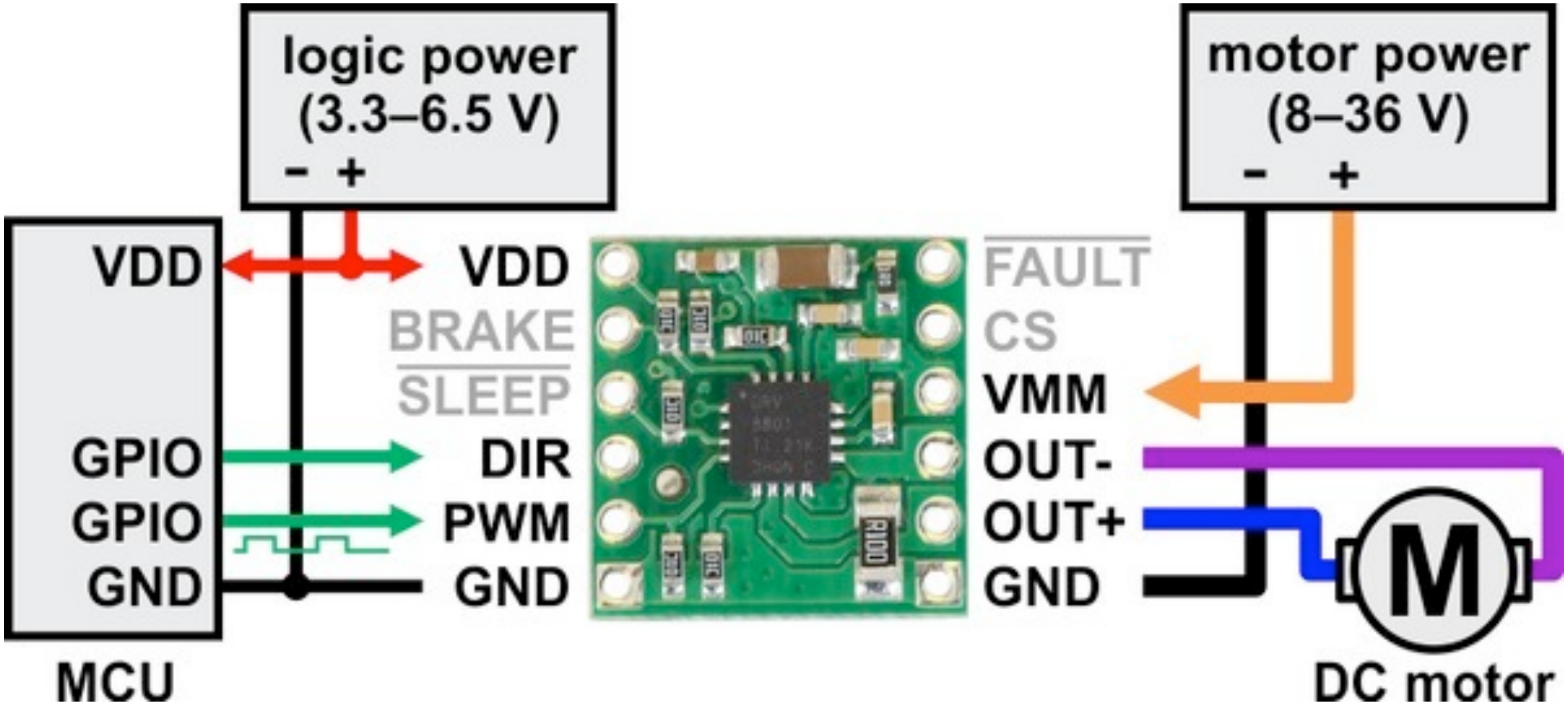


Brake



Coast

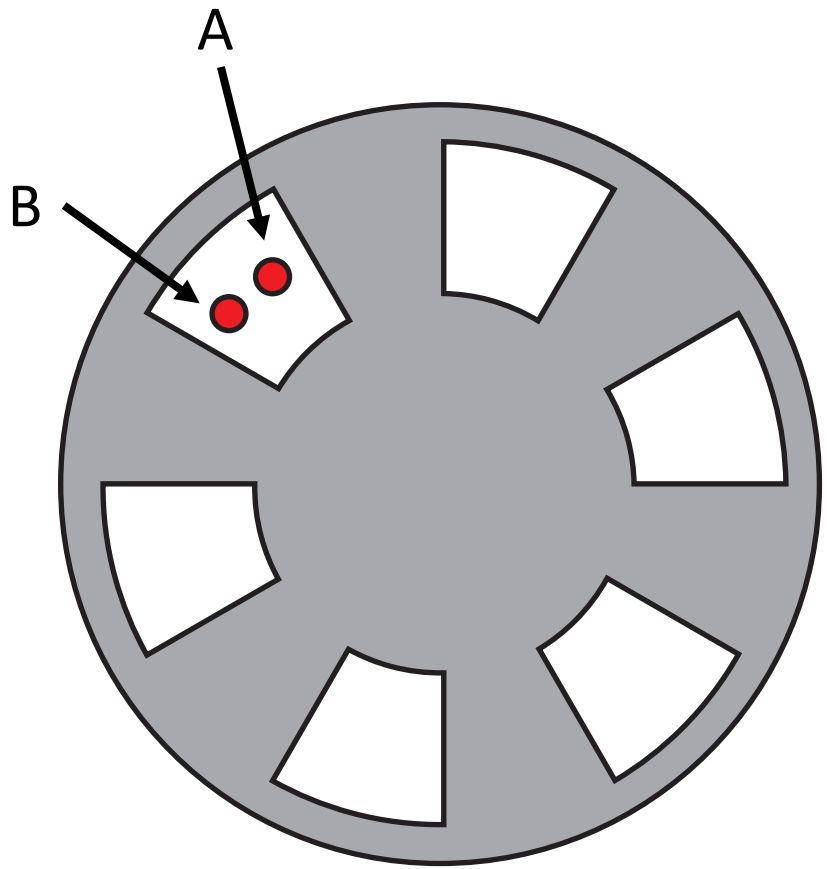
Balancebot Motor Driver

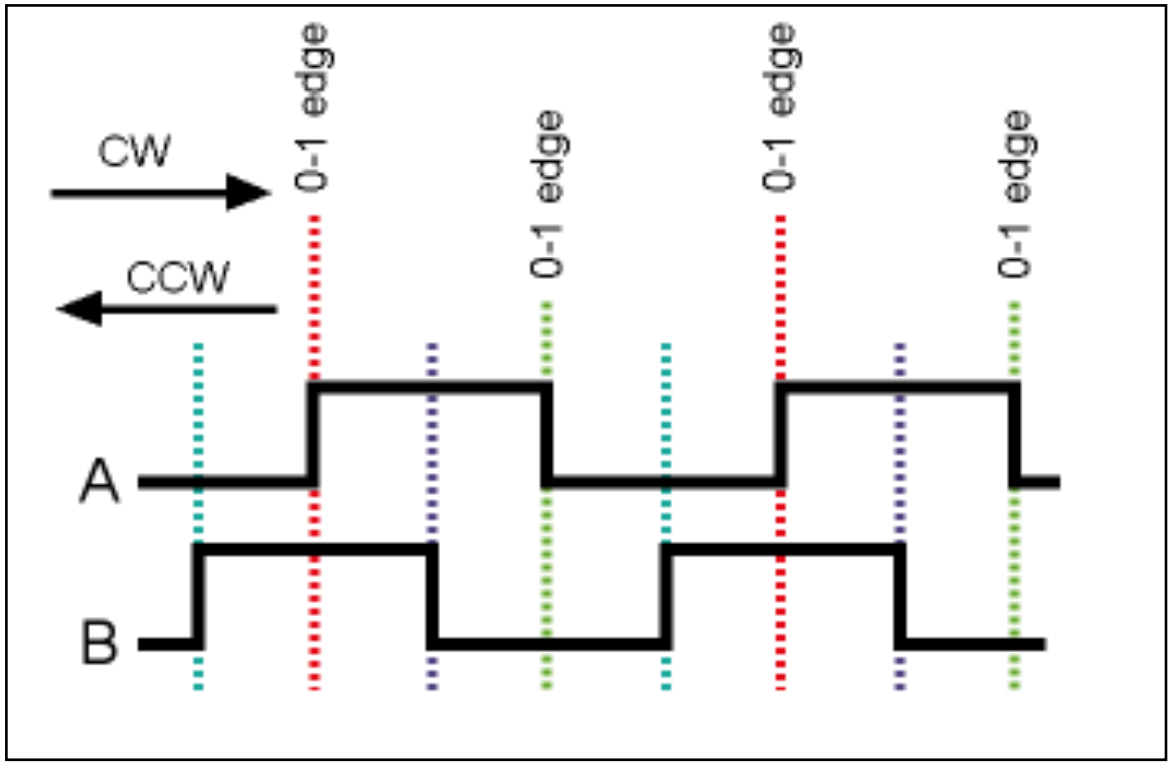


Motor Driver Module Control

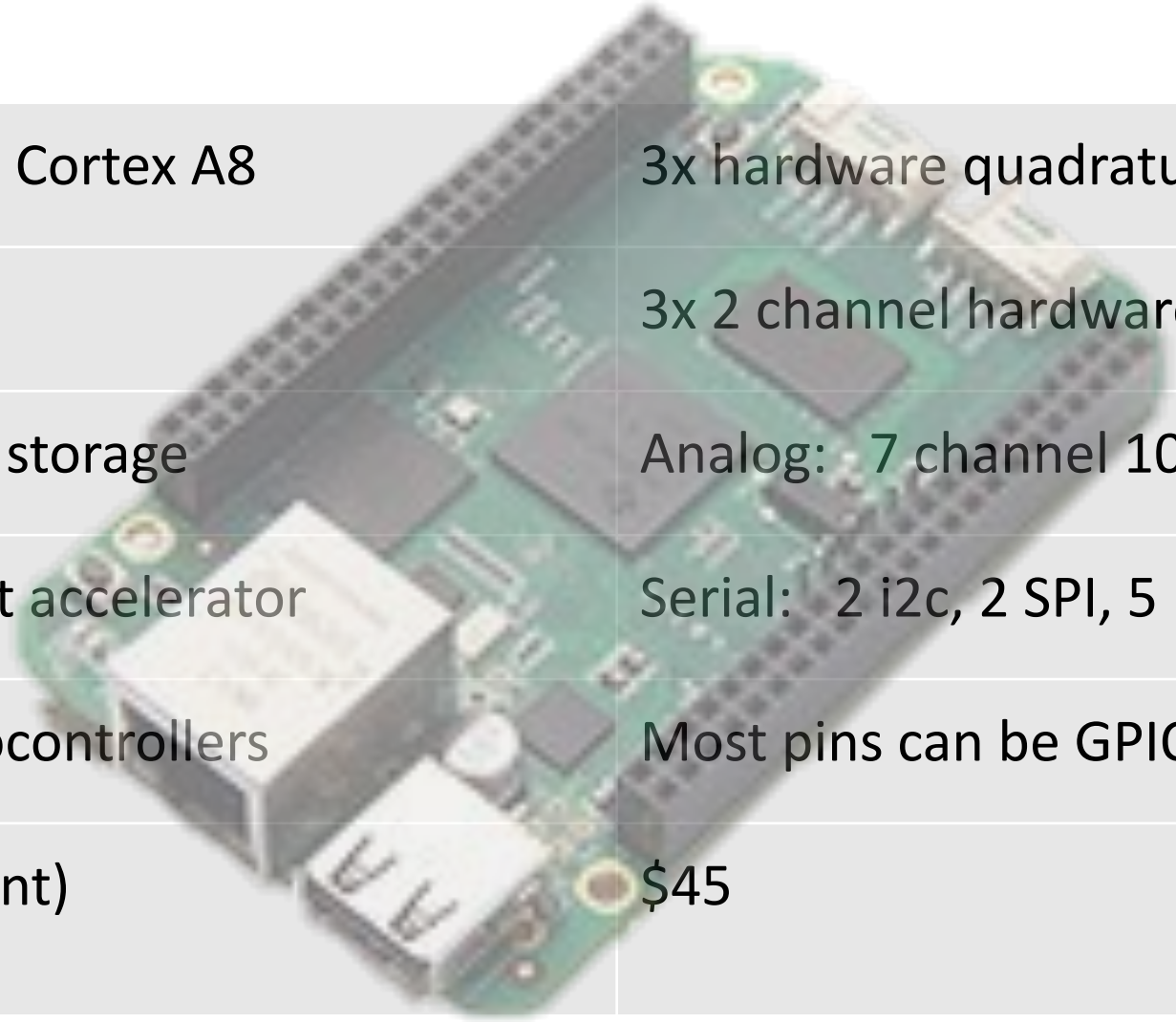
Motor Driver Module Truth Table					
PWM	DIR	BRAKE	OUT+	OUT-	operating mode
PWM	1	1	PWM (H/L)	L	forward/brake
PWM	0	1	L	PWM (H/L)	reverse/brake
L	X	1	L	L	brake low
PWM	1	0	PWM (H/OPEN)	PWM (L/OPEN)	forward/coast
PWM	0	0	PWM (L/OPEN)	PWM (H/OPEN)	reverse/coast
L	X	0	OPEN	OPEN	coast

Measuring Speed & Direction: Quadrature Encoder





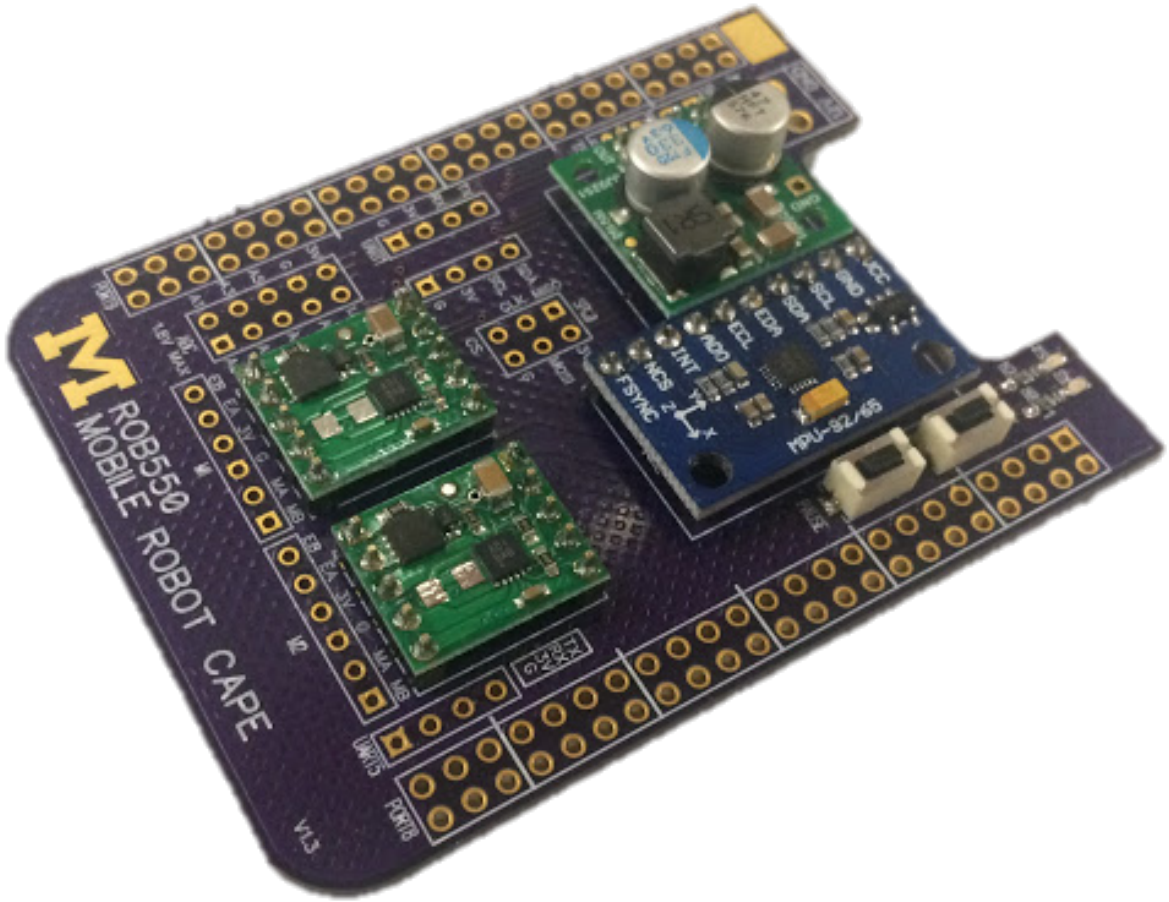
Beaglebone Green



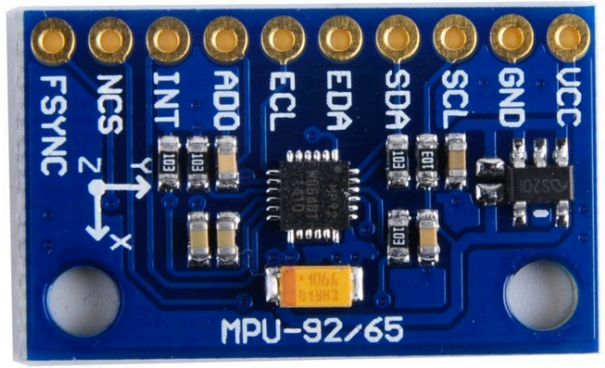
1GHz AM335x ARM Cortex A8	3x hardware quadrature decoders
512MB DDR3 RAM	3x 2 channel hardware PWMs
4GB on-board flash storage	Analog: 7 channel 10bit ADC
NEON floating-point accelerator	Serial: 2 i2c, 2 SPI, 5 UARTs
2x PRU 32-bit microcontrollers	Most pins can be GPIO
USB 2.0 (host & client) 100Mb Ethernet	$45

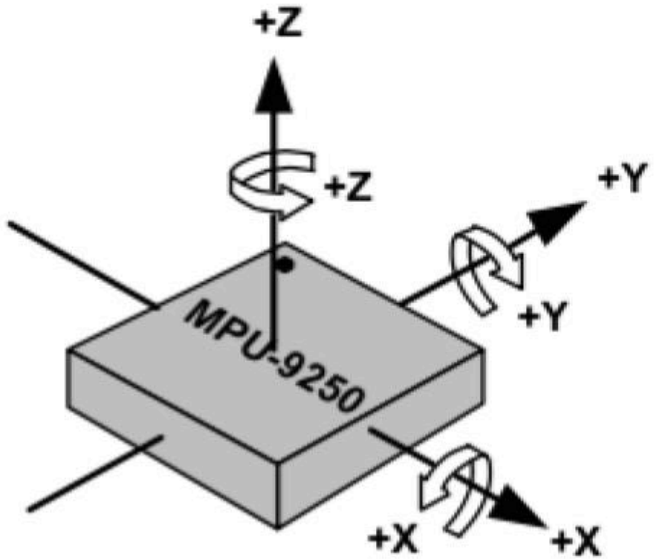
Mobile Robot Cape

- Voltage Regulator
- 2 Motor Drivers
- 2 Quadrature Decoders
- 1 IMU (gyro + accel + mag)
- 2 buttons
- 2 indicator LEDs
- 4 serial ports
  - 2 UART, i2C, SPI
- 7 ch ADC
- DSM RC receiver interface



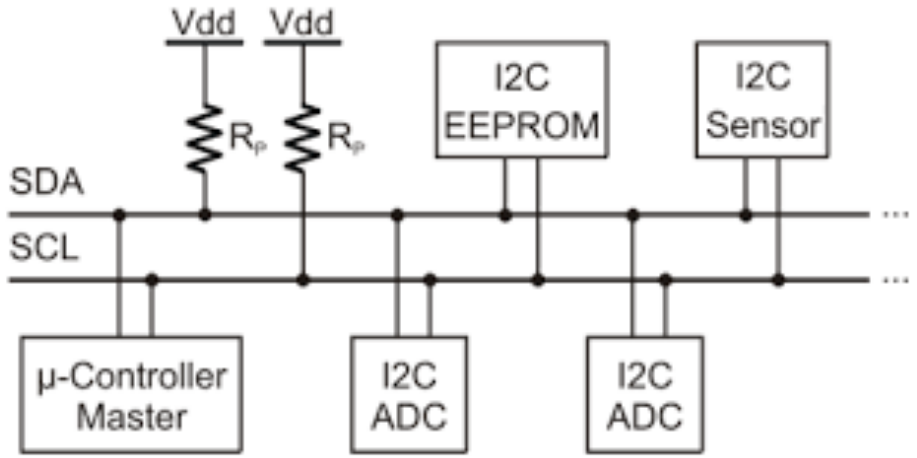
IMU – MPU9250

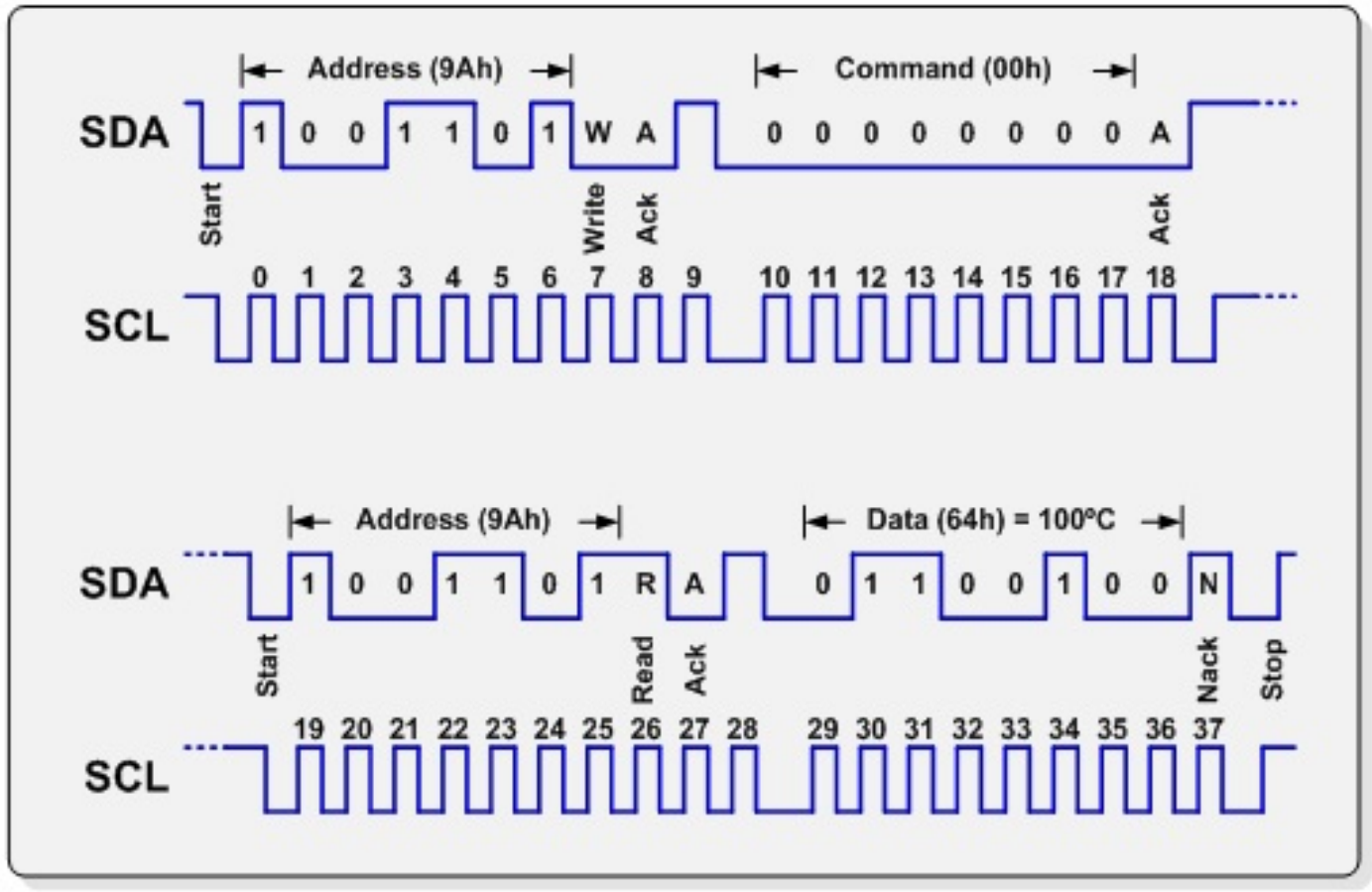




- Accelerometer, Gyroscope, Magnetometer
- Contains data-fusion processor – DMP
- DMP interrupts BB at fixed rate

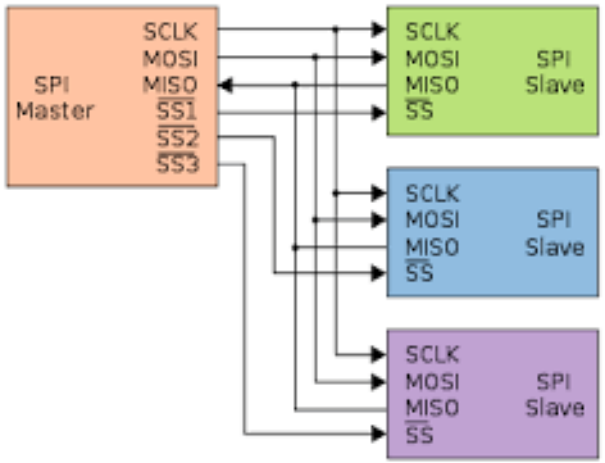
Inter-IC Communications – i2C



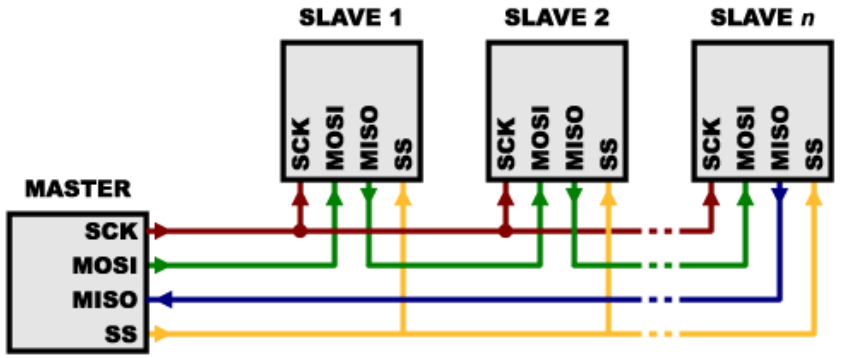


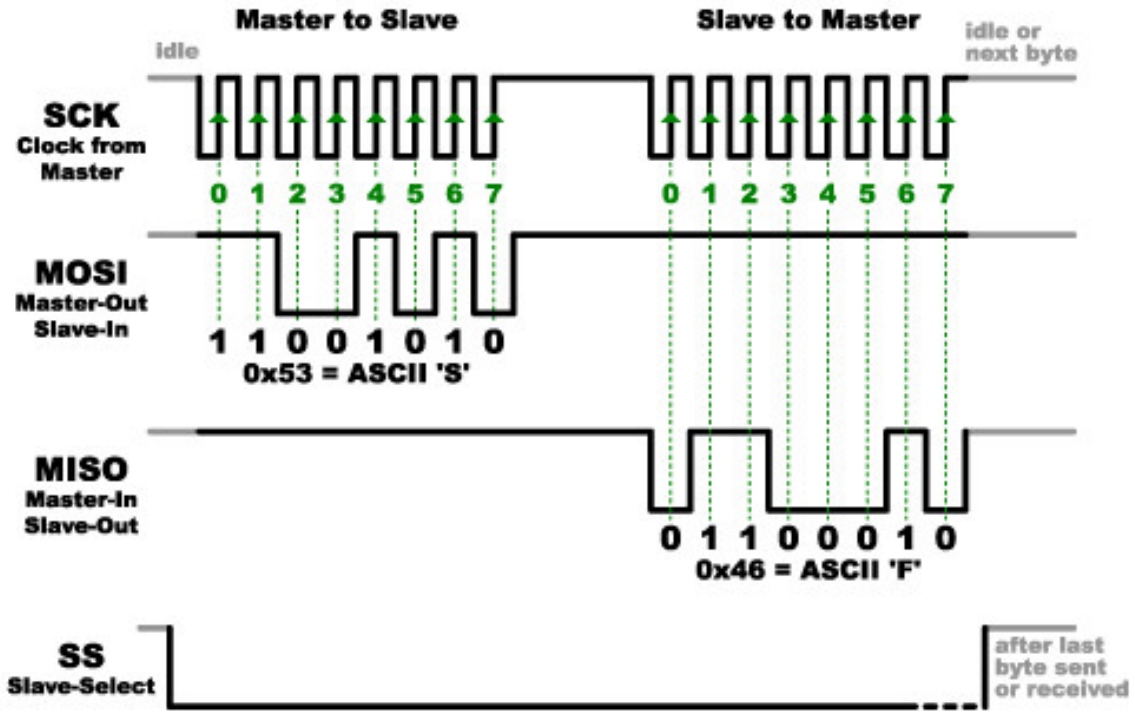
Serial Peripheral Interface - SPI

Parralel



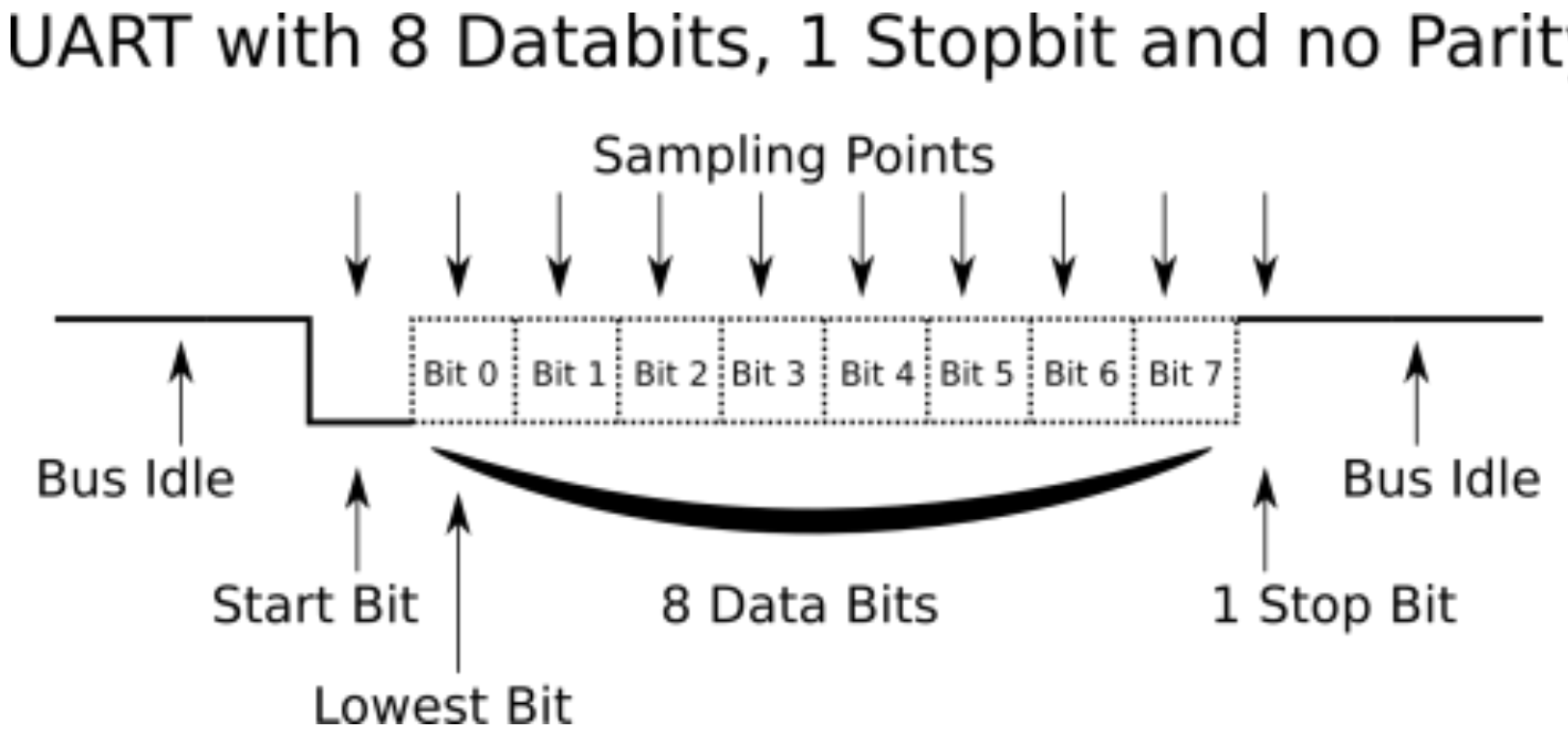
Serial





Asynchronous Serial Interface
(Universal Asynchronous Receiver Transmitter)

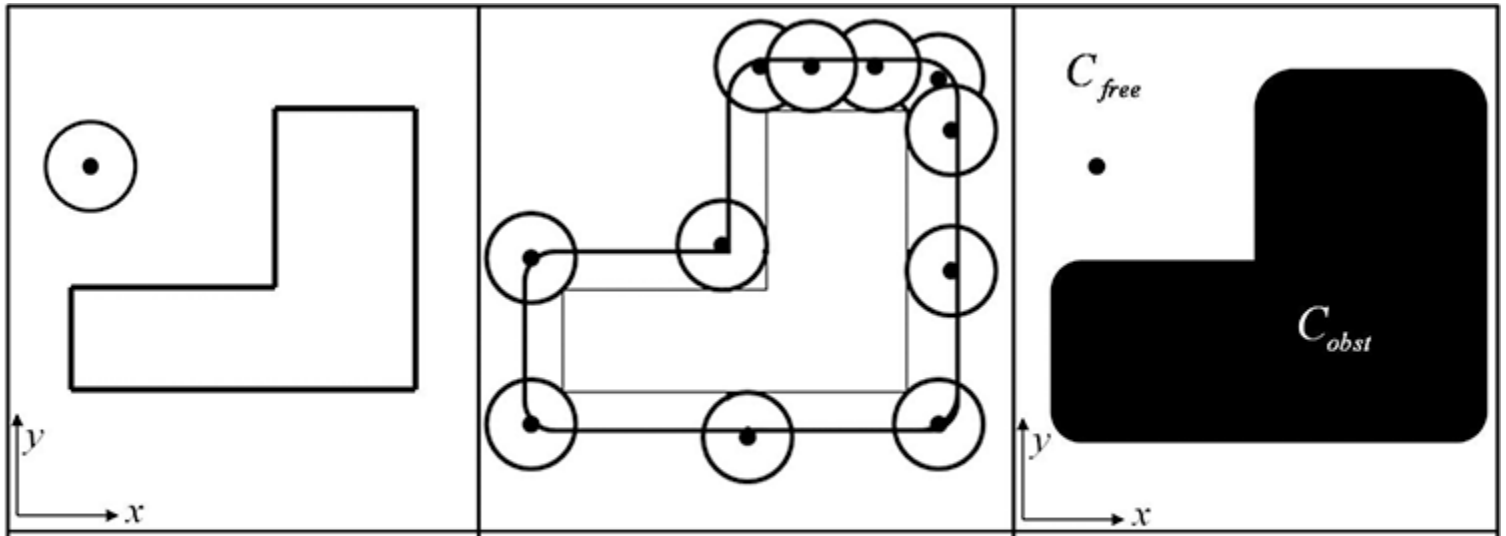
Separate TX and RX pins, fixed baud rate



Path Planning

Configuration Space

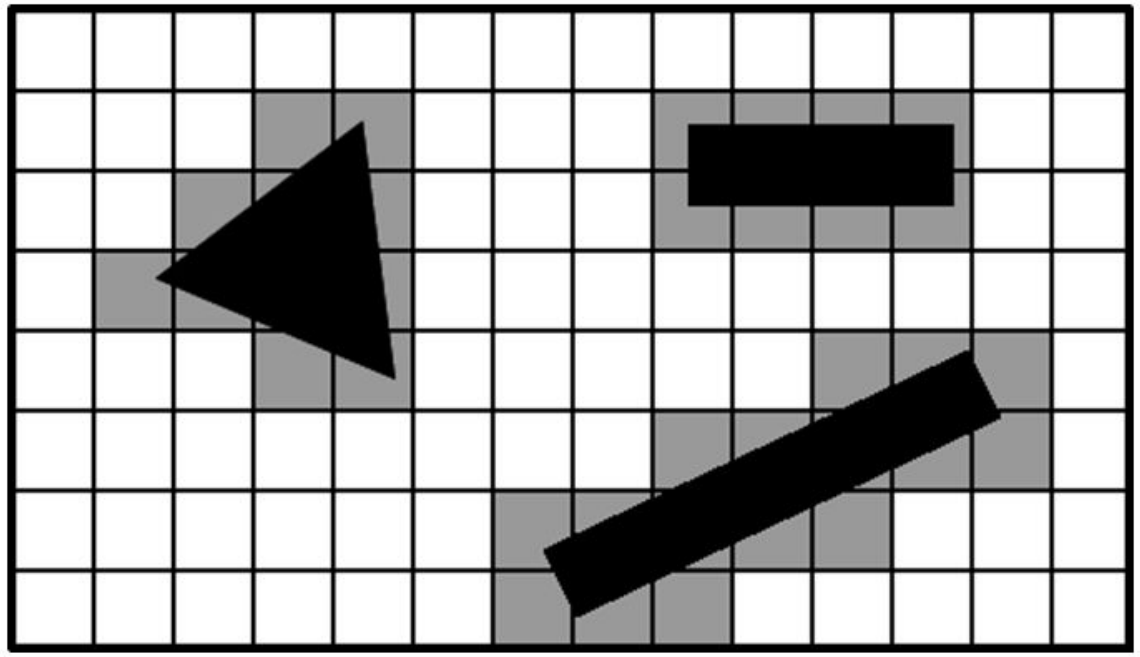
- Portion of space the robot can occupy.
- Reduces robot to a point.

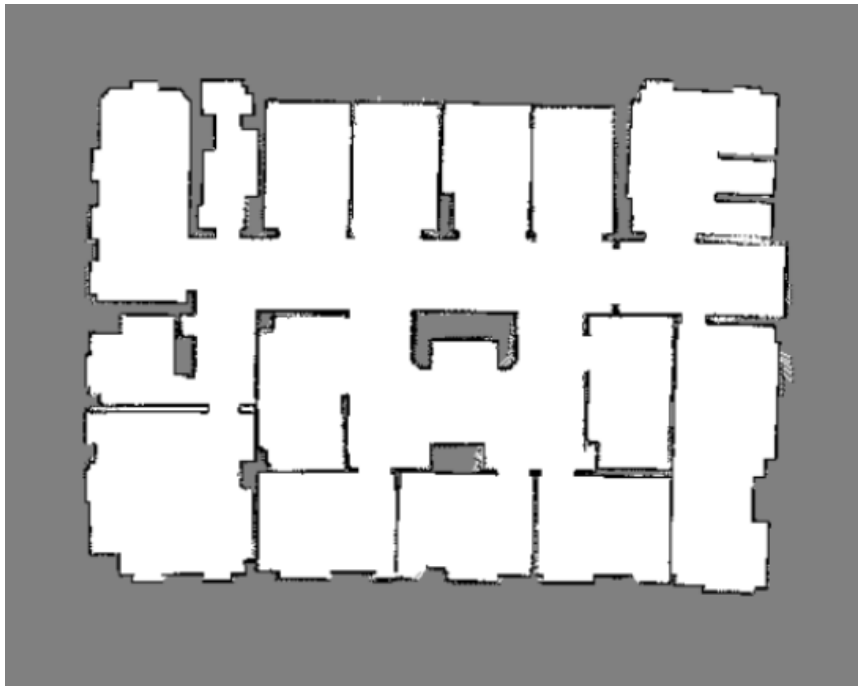


Mobile robot workspace. | The process of constructing the C-space. | Robot C-space.

Occupancy Grid

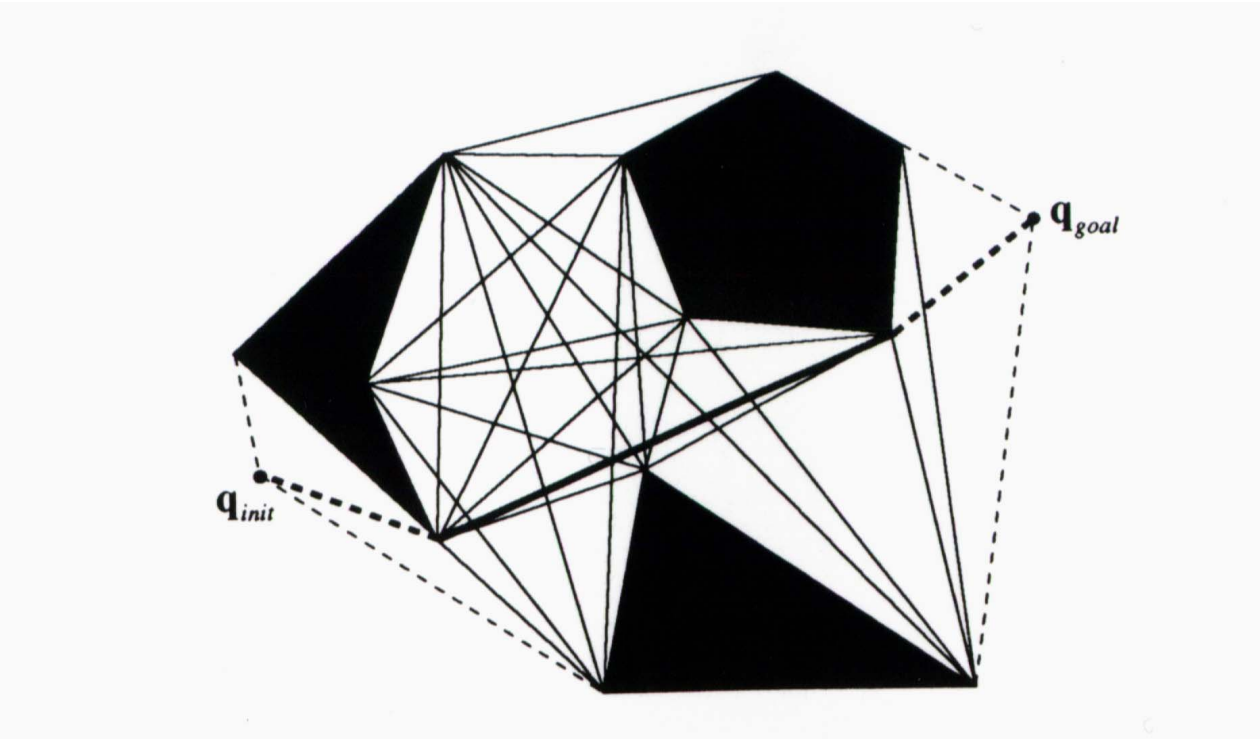
- Represent space as a grid of occupied, or free cells





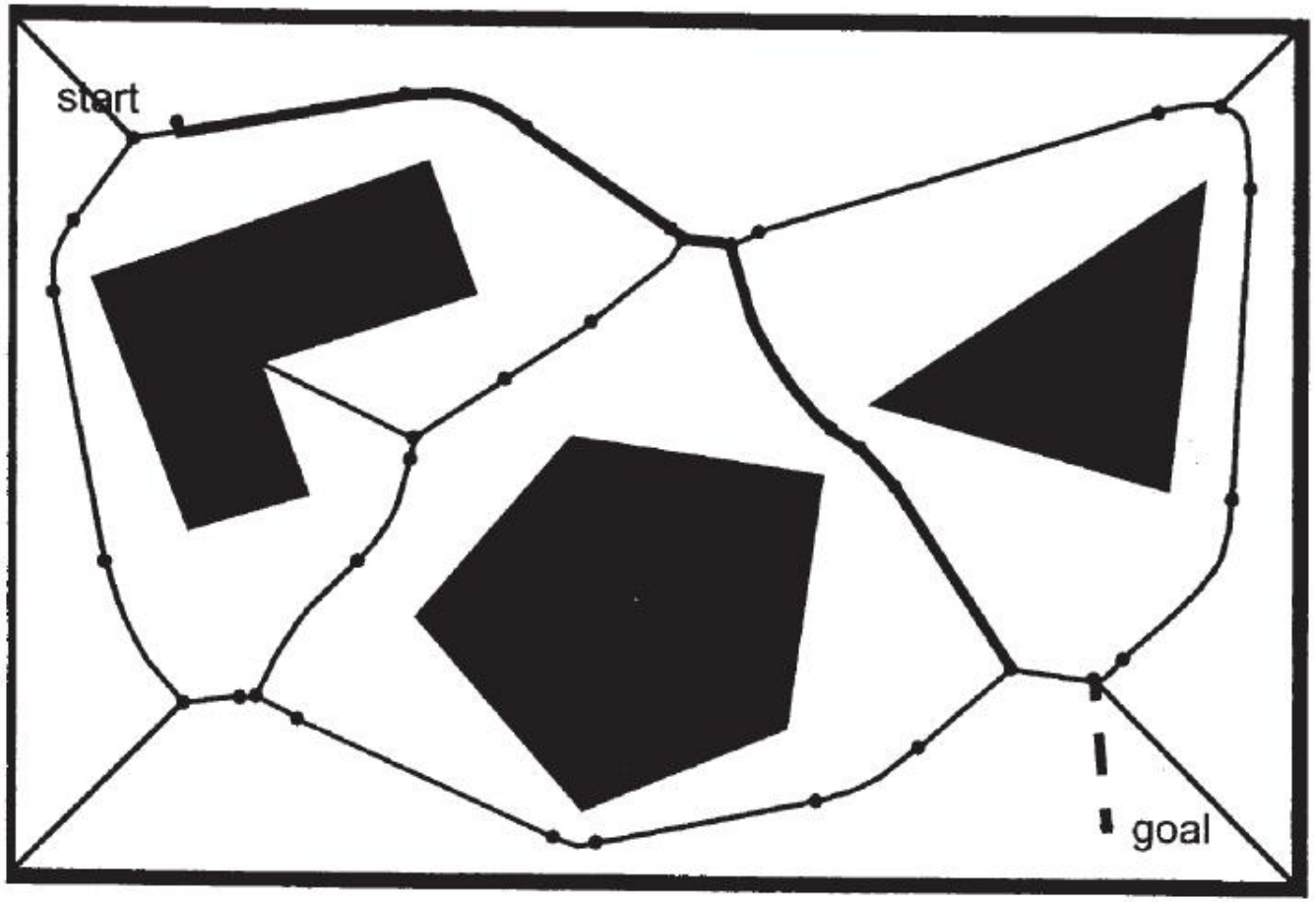
Visibility Graph

- Make obstacles in C-space polygons
- Set node at each vertex
- Set node at start and goal
- Edges between nodes based on visibility
- Must use collision detection to build edges



Voronoi Graph

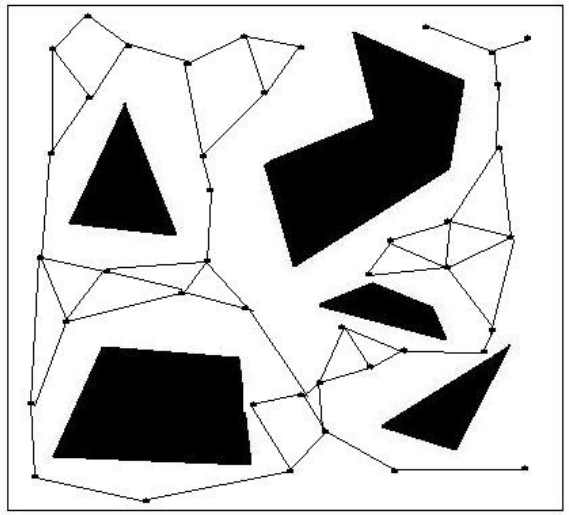
- Paths are points where distance to the two nearest obstacles are the same
- Paths favor moving the robot through the middle of large open spaces

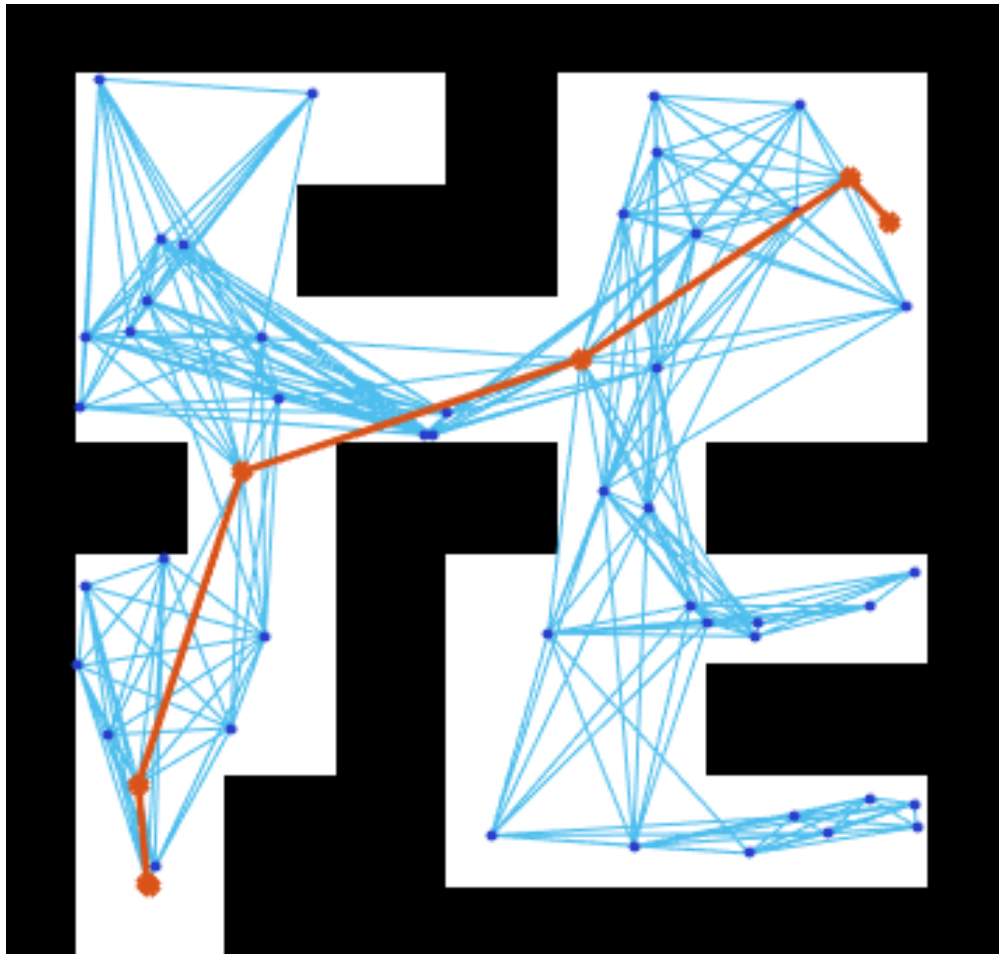


Voronoi Graph

Probabilistic Roadmap (PRM)

- Randomly sample free points in C-space to generate nodes
- Connect edges to visible nodes within a horizon





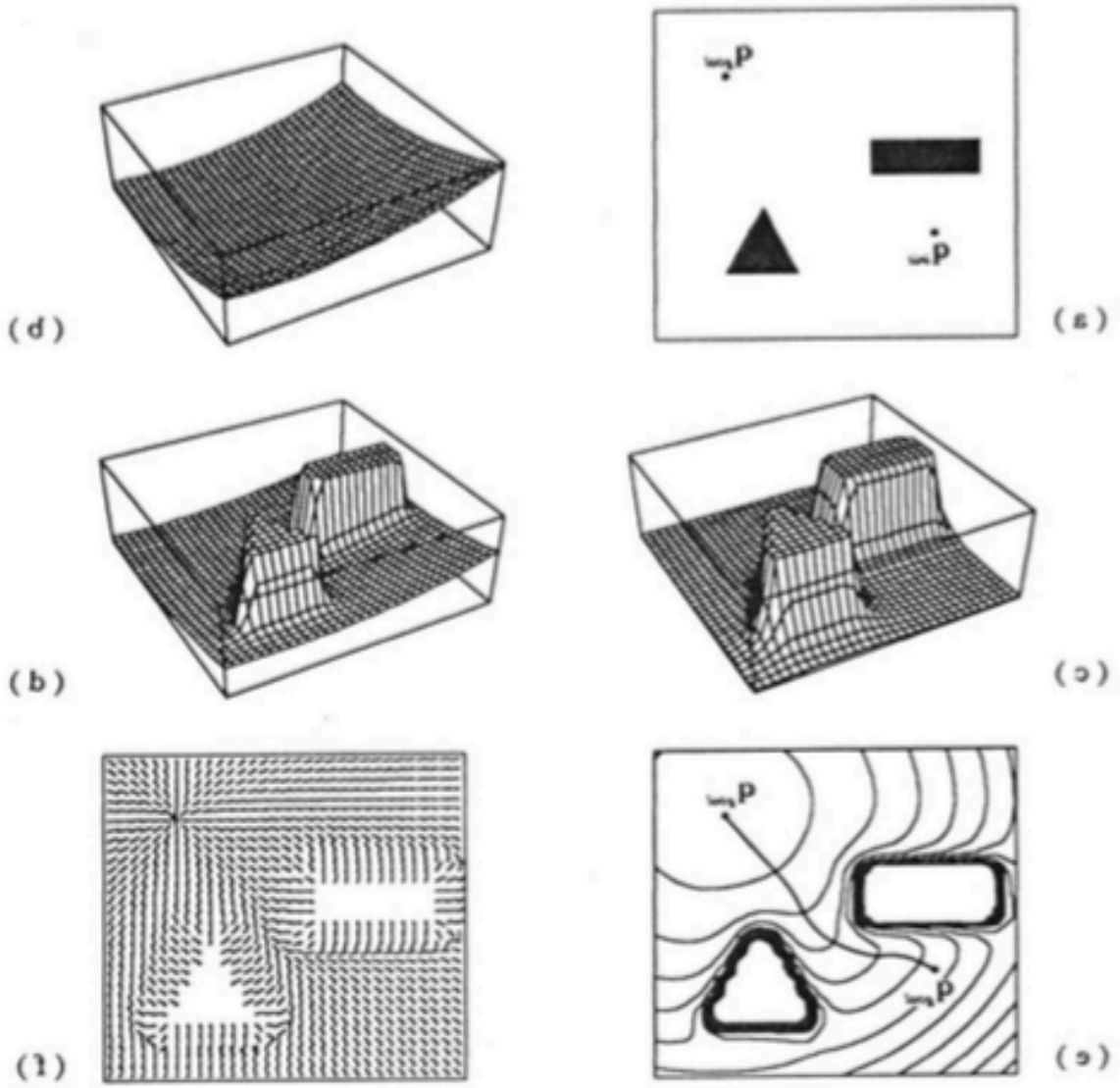
Rapidly Exploring Random Tree (RRT)

- Randomly explore space, biased towards the goal
- Tree will eventually fill space efficiently
- Can grow from start and goal and stop when trees meet

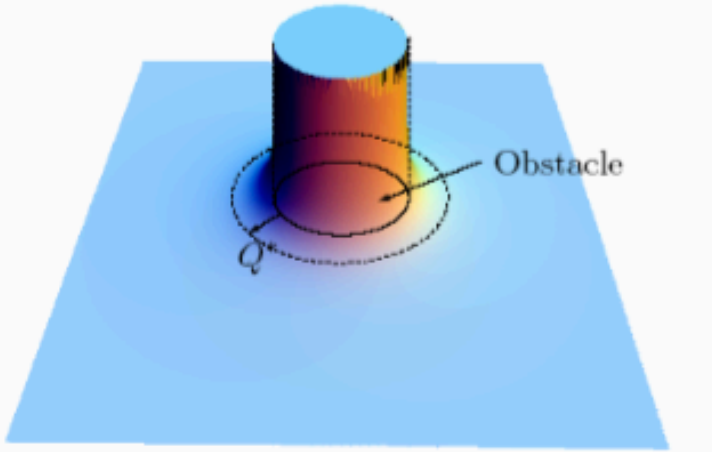
Potential Field Planning

$$U(q) = U_{\text{att}}(q) + U_{\text{rep}}(q)$$

$$F(q) = -\nabla U(q)$$



Repulsive Potential for Obstacles



$$U_{\text{rep}}(q) = \begin{cases} \frac{1}{2}\eta(\frac{1}{D(q)} - \frac{1}{Q^*})^2, & D(q) \le Q^*, \\ 0, & D(q) > Q^*, \end{cases}$$

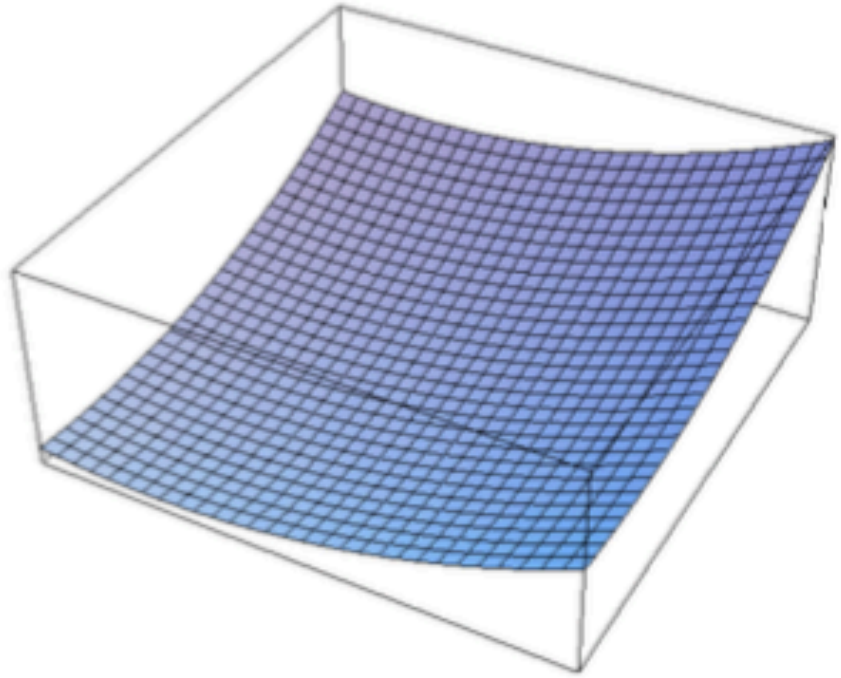
whose gradient is

$$\nabla U_{\text{rep}}(q) = \begin{cases} \eta\left(\frac{1}{Q^*} - \frac{1}{D(q)}\right)\frac{1}{D^2(q)}\nabla D(q), & D(q) \le Q^*, \\ 0, & D(q) > Q^*, \end{cases}$$

Attractive Potential for Goal

$$U_{\text{att}}(q) = \begin{cases} \frac{1}{2}\zeta d^2(q, q_{\text{goal}}), & d(q, q_{\text{goal}}) \leq d^*_{\text{goal}}, \\ d^*_{\text{goal}}\zeta d(q, q_{\text{goal}}) - \frac{1}{2}\zeta (d^*_{\text{goal}})^2, & d(q, q_{\text{goal}}) > d^*_{\text{goal}}. \end{cases}$$

$$\nabla U_{\text{att}}(q) = \begin{cases} \zeta(q - q_{\text{goal}}), & d(q, q_{\text{goal}}) \leq d^*_{\text{goal}}, \\ \frac{d^*_{\text{goal}}\zeta(q - q_{\text{goal}})}{d(q, q_{\text{goal}})}, & d(q, q_{\text{goal}}) > d^*_{\text{goal}}, \end{cases}$$

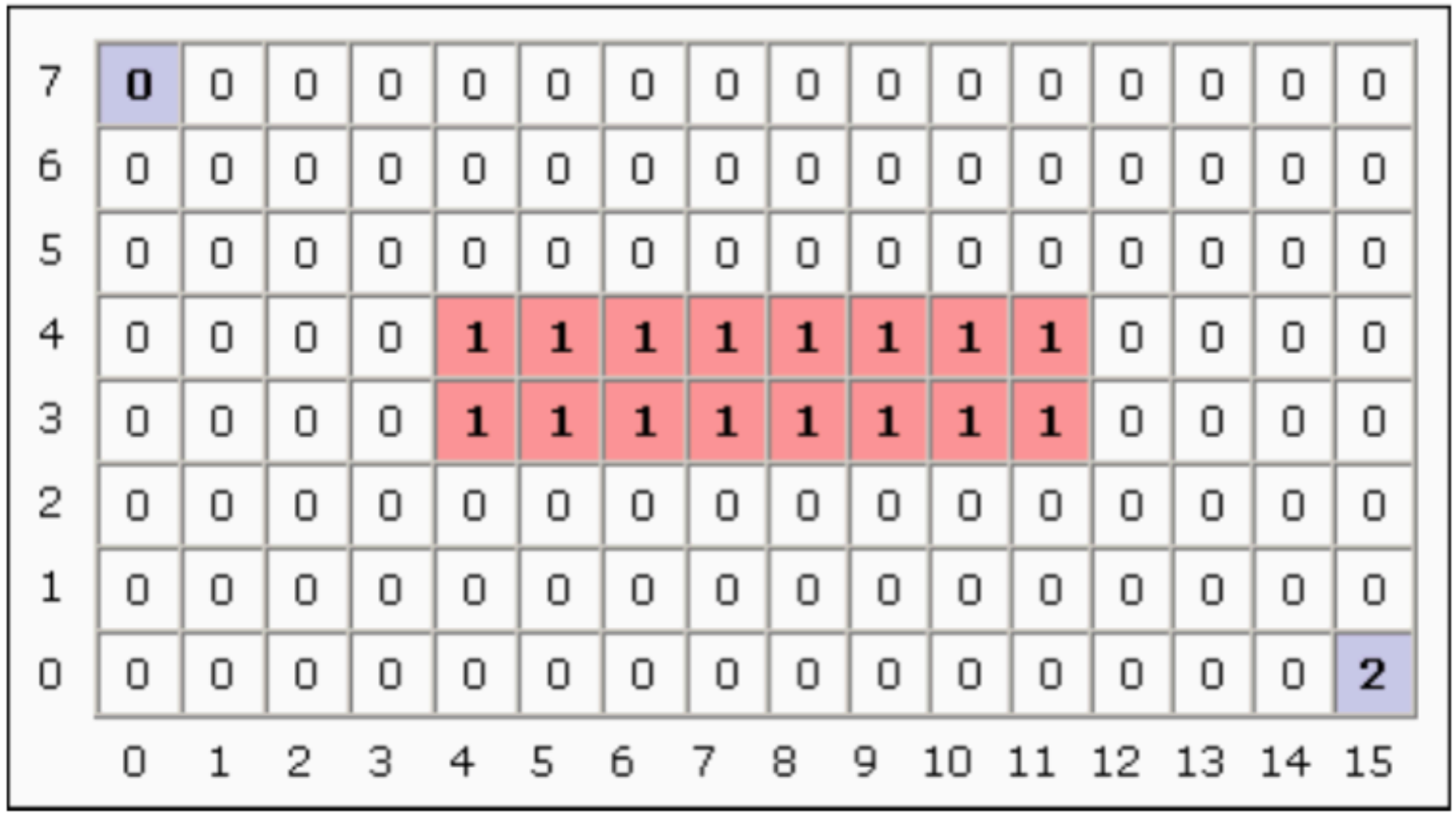


Problems with Potential Fields – Local Minima

- Critical points (where $\nabla U(q) = 0$) are problems
- Represent local maxima, minima and saddle points.
- Can use a random walk to get out.
- Navigation Functions are special potential field constructions with no local minima, only one minimum point for the goal
- A Wavefront planner is a simple potential field planner on a grid, is not optimal, but avoids these critical points.

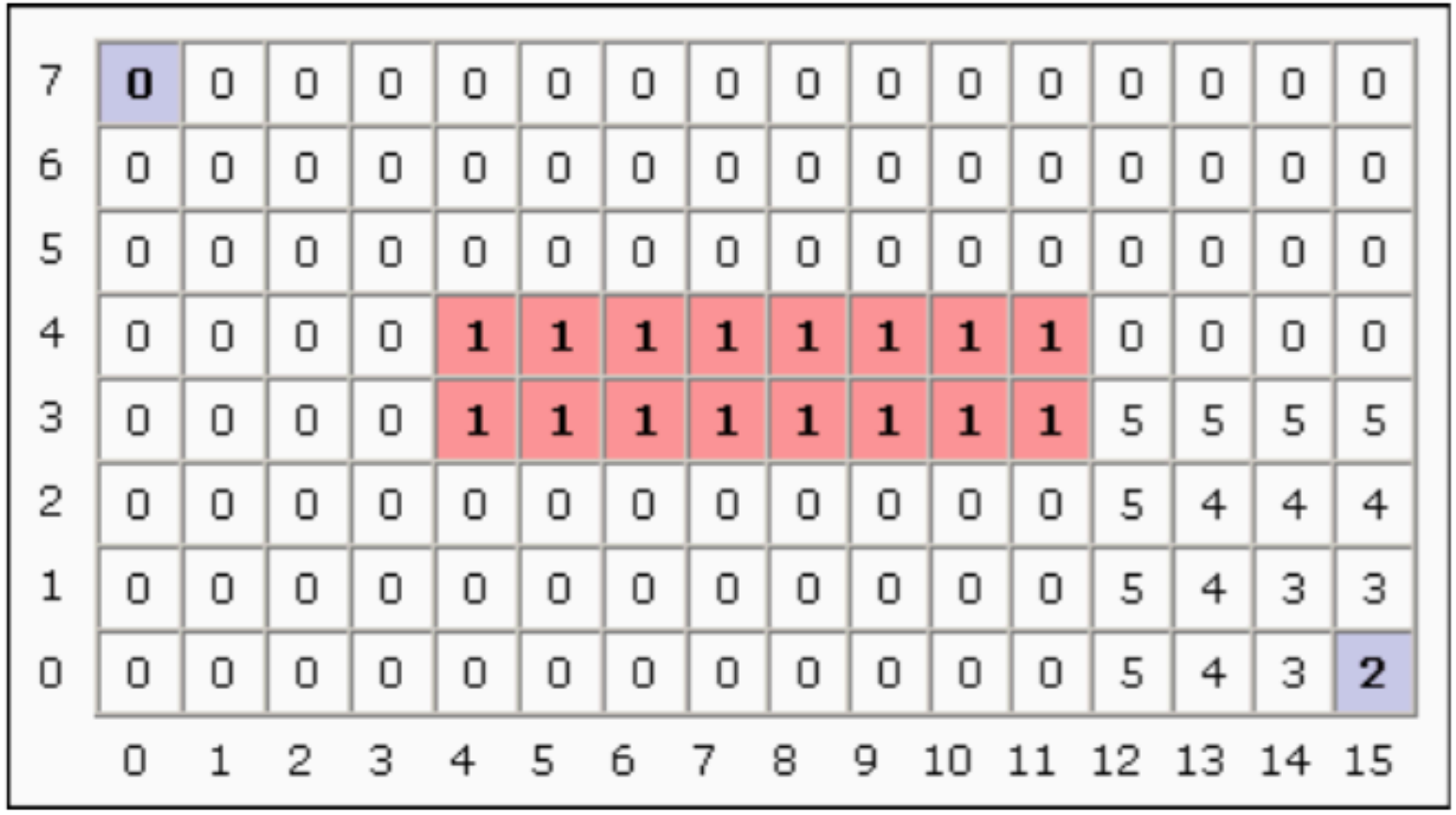
Wavefront Planner

- Assign obstacles in occupancy grid a value of 1, and goal a value of 2.



Wavefront Planner

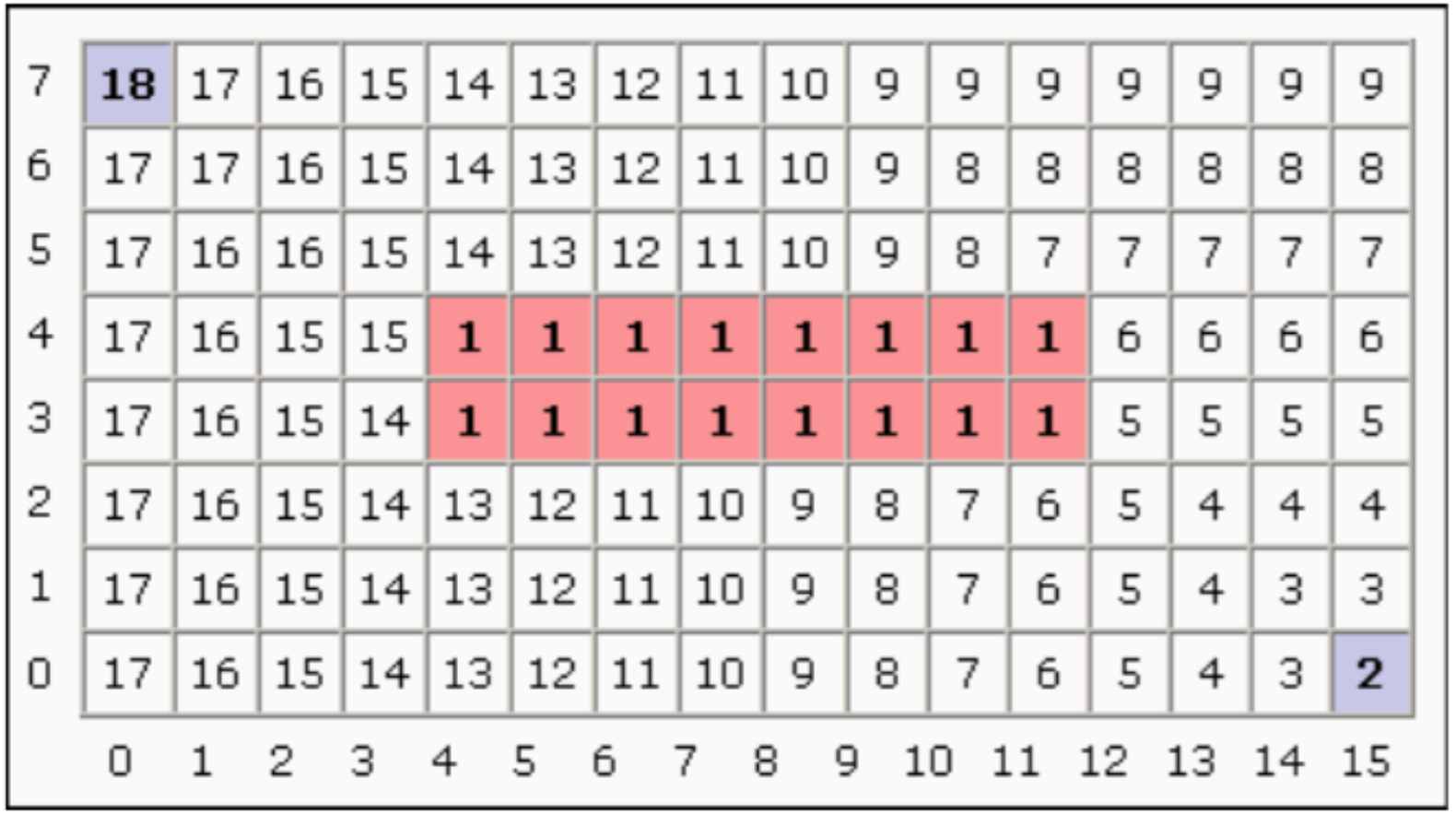
- Starting with the goal set adjacent cells to +1 and repeat…



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
7	**0**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	0	0	0	0
3	0	0	0	0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	5	5	5	5
2	0	0	0	0	0	0	0	0	0	0	0	0	5	4	4	4
1	0	0	0	0	0	0	0	0	0	0	0	0	5	4	3	3
0	0	0	0	0	0	0	0	0	0	0	0	0	5	4	3	**2**

Wavefront Planner

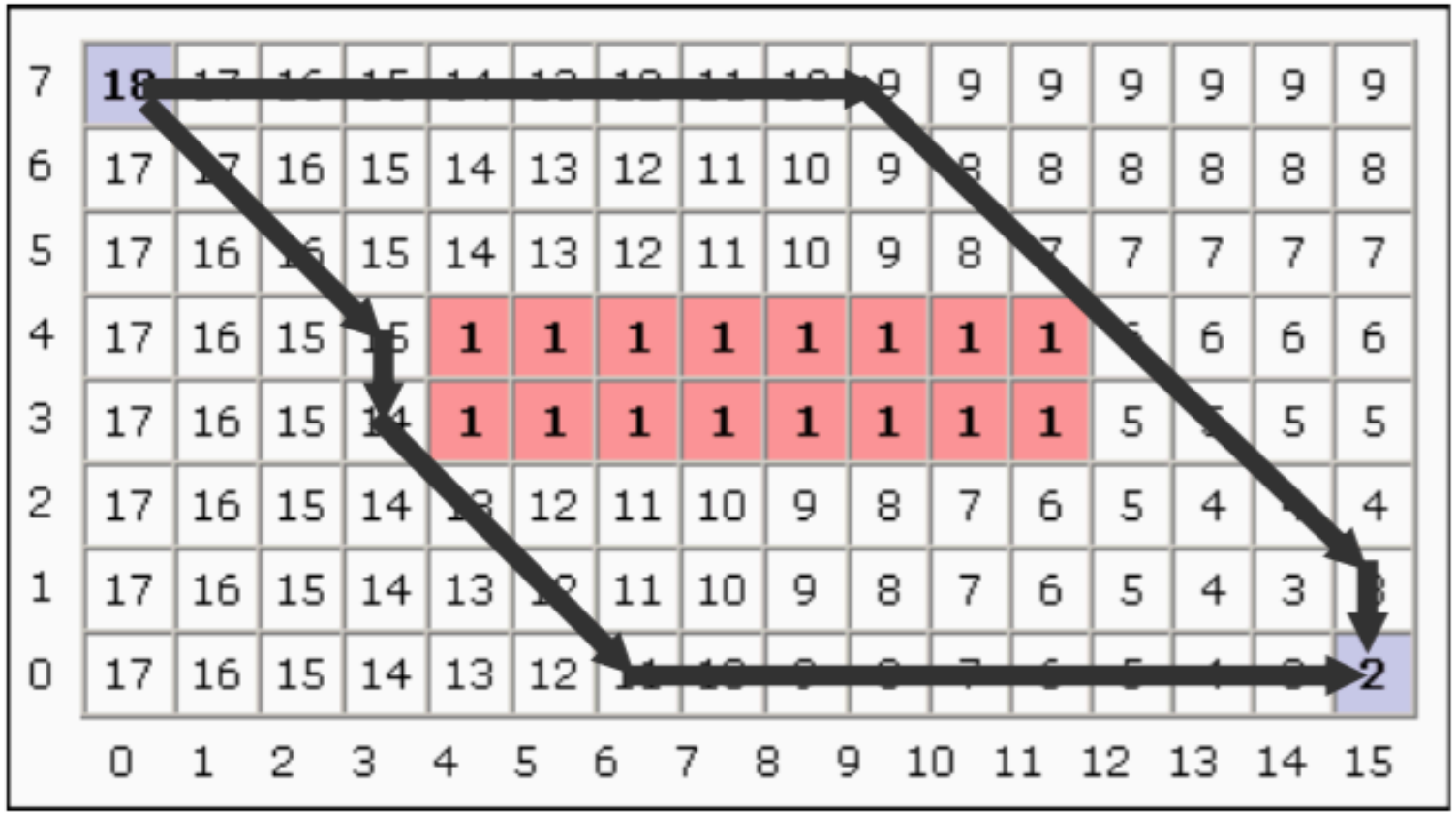
- When complete start cell has a number.
- only unnavigable areas should be 0.



7	**18**	17	16	15	14	13	12	11	10	9	9	9	9	9	9	9
6	17	17	16	15	14	13	12	11	10	9	8	8	8	8	8	8
5	17	16	16	15	14	13	12	11	10	9	8	7	7	7	7	7
4	17	16	15	15	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	6	6	6	6
3	17	16	15	14	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	5	5	5	5
2	17	16	15	14	13	12	11	10	9	8	7	6	5	4	4	4
1	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	3
0	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	**2**
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Wavefront Planner

- Shortest path is found by always moving to a cell with a lower value.



Finding a path - Graph Search

- Breadth First Search
- Depth First Search
- A*
- Dijkstra’s Algorithm
- Greedy search
- Dijkstra’s and Greedy are two special cases of A*

…to the chalk board!

Grid Search A* Movie…

https://www.youtube.com/watch?v=-L-WgKMFuhE

Basic Probability

Probabilistic Robotics CH2.2

Random Variables

- Random variables can take on any value
- Variables can be discrete i.e. coin toss (heads|tails)
- Variables can be continuous i.e. range finder measurement in meters
- The probability a random variable has a certain value is

$$p(X = x)$$

- Example -> coin toss:

$$p(X = \text{heads}) = p(X = \text{tails}) = \frac{1}{2}$$

Probability Density Functions

- Probabilities of discrete variables sum to one: $\sum_x p(X = x) = 1$
- Probabilities of continuous variables are a probability density function whose integral over all values is $\int p(x)dx = 1$
- Common P.D.F. is the Gaussian or Normal distribution with mean μ and variance σ^2

$$p(x) = (2\pi\sigma^2)^{-\frac{1}{2}} \exp\left\{ -\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2} \right\}$$

- We commonly write this specifying the variable, mean and variance:

$$\mathcal{N}(x; \mu, \sigma^2)$$

Joint Probability

- The joint probability of 2 variables is $p(x, y) = p(X = x \ \& \ Y = y)$
- If these are independent $p(x, y) = p(x)p(y)$
- Conditional probability: probability of one variable given known value of another variable

$$p(x|y) = p(X = x \mid Y = y)$$

- If two variable are independent, $p(x|y) = p(x)$
- Chain rule

$$p(x, y) = p(x|y)p(y) = p(y|x)p(x)$$

Bayes Rule

- Bayes rule allows us to infer the conditional probability of A given B, based on the inverse conditional probability of B given A, and the independent probabilities of A and B.

$$p(A|B) = \frac{p(B|A)p(A)}{p(B)}$$

Bayes' Rule

$$p(A|B) = \frac{p(B|A)p(A)}{p(B)}$$

- If *A* is a quantity we would like to infer from data *B*:
- *p*(*A*) is the *prior* probability of A
- *p*(*A*|*B*) is the *posterior* probability after taking data *B* into account
- *p*(*B*|*A*) is the *likelihood* of the data given the hypothesis
- *p*(*B*) is the *prior* probability of the data and effectively normalizes the posterior to one.

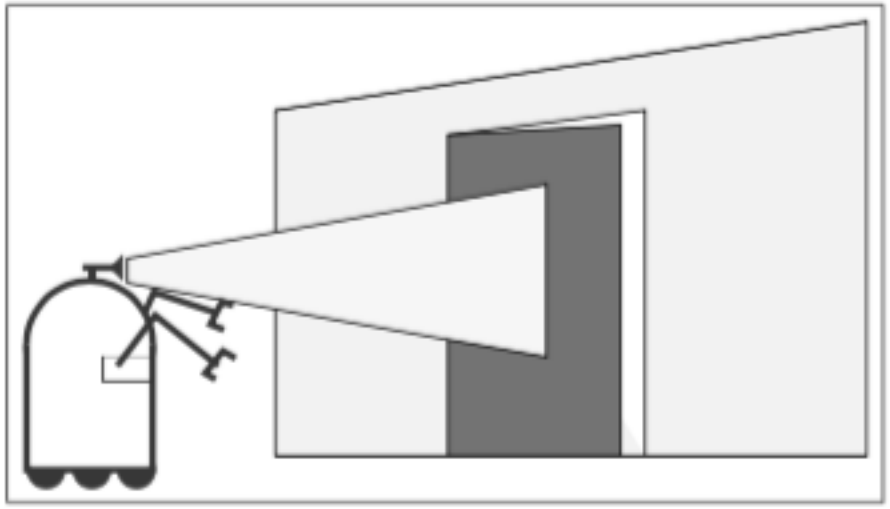
Why is Bayes' Rule so useful

$$p(A|B) = \frac{p(B|A)p(A)}{p(B)}$$

- Diagnostic evidence *p*(*disease* | *symptom*) is often hard to get but what you want to know.
- Causal evidence *p*(*symptom* | *disease*) is often easier to get.
- *p*(*disease*) is easy to get
- *p*(*symptom*) is just a normalizer

Door Sensor Example

- Your robot has a noisy discrete sensor for detecting if a door is open or closed.
- If the door is open, then 70% of the time the sensor reads HIGH. But 30% of the time it reads LOW.
- If the door is closed, then 90% of the time the sensor reads LOW. But 10% of the time it reads HIGH.



Door Sensor Example

- Your sensor gives you 5 consecutive measurements:
  - HIGH, LOW, HIGH, LOW, HIGH
- What do you conclude?
  - 1. The sensor is worthless
  - 2. It is likely the door is closed
  - 3. It is slightly more likely than not door is closed
  - 4. It is much more likely that the door is closed

Door Sensor Example

We have these pieces of information:

$$p(H|O) = 0.7$$
$$p(L|O) = 0.3$$
$$p(H|C) = 0.1$$
$$p(L|C) = 0.9$$
$$p(O) = 0.5$$
$$p(C) = 0.5$$

What is the probability your sensor would read $HLHLH$ if the door is open?

What is the probability your sensor would read $HLHLH$ if the door is closed?

Door Sensor Example

We have these pieces of information:

$p(H|O) = 0.7$
$p(L|O) = 0.3$
$p(H|C) = 0.1$
$p(L|C) = 0.9$
$p(O) = 0.5$
$p(C) = 0.5$

What is the probability your sensor would read $HLHLH$ if the door is open?

What is the probability your sensor would read $HLHLH$ if the door is closed?

$p(HLHLH|O) = 0.7 * 0.3 * 0.7 * 0.3 * 0.7 = 0.03087$
$p(HLHLH|C) = 0.1 * 0.9 * 0.1 * 0.9 * 0.1 = 0.00081$

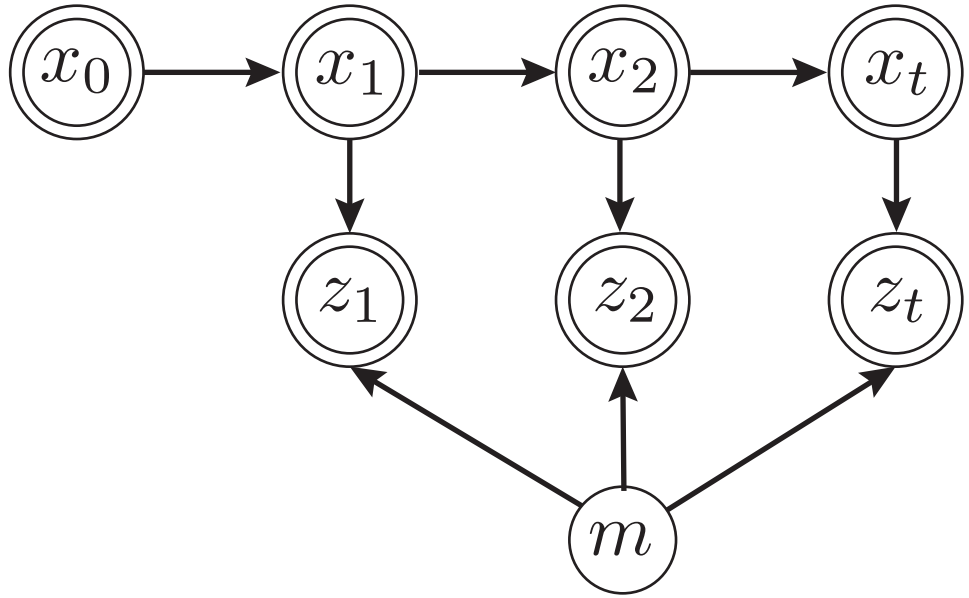
Door Sensor Example

- Let S equal the sensor sequence $HLHLH$.
- What is the probability of the door being open given the sequence S $p(O|S)$?

$$
\begin{aligned}
p(O|S) &= \frac{p(S|O)p(O)}{p(S)} = \frac{p(S|O)p(O)}{p(S|O)p(O) + p(S|C)p(C)} \\
&= \frac{(0.7 \cdot 0.3 \cdot 0.7 \cdot 0.3 \cdot 0.7)(0.5)}{(0.7 \cdot 0.3 \cdot 0.7 \cdot 0.3 \cdot 0.7)(0.5) + (0.1 \cdot 0.9 \cdot 0.1 \cdot 0.9 \cdot 0.1)(0.5)} \\
&= \frac{343}{352} \approx 0.974
\end{aligned}
$$

Using Conditional Probability to make a map

- Given robot poses $x_{1:t}$ and laser rangefinder measurements $z_{1:t}$ infer a map of the environment
- Expressed probabilistically as $p(m|z_{1:t}, x_{1:t})$



- Double circle is known variable
- Arrow indicate generation
- Single circle is latent variable

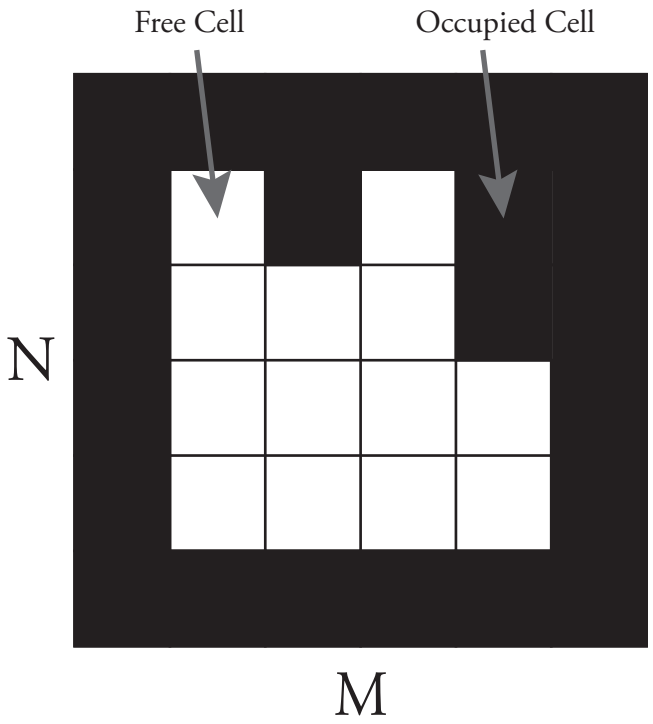
Occupancy Grid Mapping

- Map is a M x N matrix of cells
- Cell is either occupied or unoccupied
- Probability $p(m_i)$ is probability cell is occupied

$$p(m|x_{1:t}, z_{1:t}) = \prod_i p(m_i|x_{1:t}, z_{1:t})$$

- Map can be inferred from a Bayes filter with a static state

$$m = \{m_i\}_{M \times N}$$



Odds Ratio & Log Odds

- A is binary state *occ(i,j)* and *B* is sensor reading
- **Probability** a cell is occupied *p(occ(i,j))* = *p(A)* has range [0,1]
- Probability a cell is free *p(¬A)*
- **Odds** of being occupied *o(occ(i,j))* = *p(A)/p(¬A)* has range [0,∞]
- **Log odds** log *o(occ(i,j))* has range [-∞,∞]
- Each cell *C(i,j)* holds the value of log *o(occ(i,j))*
- *C(i,j)* = 0 corresponds to *p(occ(i,j))* = 0.5

Bayes' Law using Odds

- Bayes' Law: $$p(A|B) = \frac{p(B|A)p(A)}{p(B)}$$
- Likewise: $$p(\neg A|B) = \frac{p(B|\neg A)p(\neg A)}{p(B)}$$
- So: $$o(A|B) = \frac{p(A|B)}{p(\neg A|B)} = \frac{p(B|A)p(A)}{p(B|\neg A)p(\neg A)}$$
$$= \lambda(B|A)o(A)$$
- Where: $$\lambda(B|A) = \frac{p(B|A)}{p(B|\neg A)}$$

Updating the map using Bayes' Law

- Bayes' Law can be written as

$$o(A|B) = \lambda(B|A)o(A)$$

posterior — Sensor update — prior

- Take log odds to make multiplication into addition

$$\log o(A|B) = \log \lambda(B|A) + \log o(A)$$

- For each cell take add the evidence $\log \lambda(B|A)$ into the cell

Map Update Algorithm

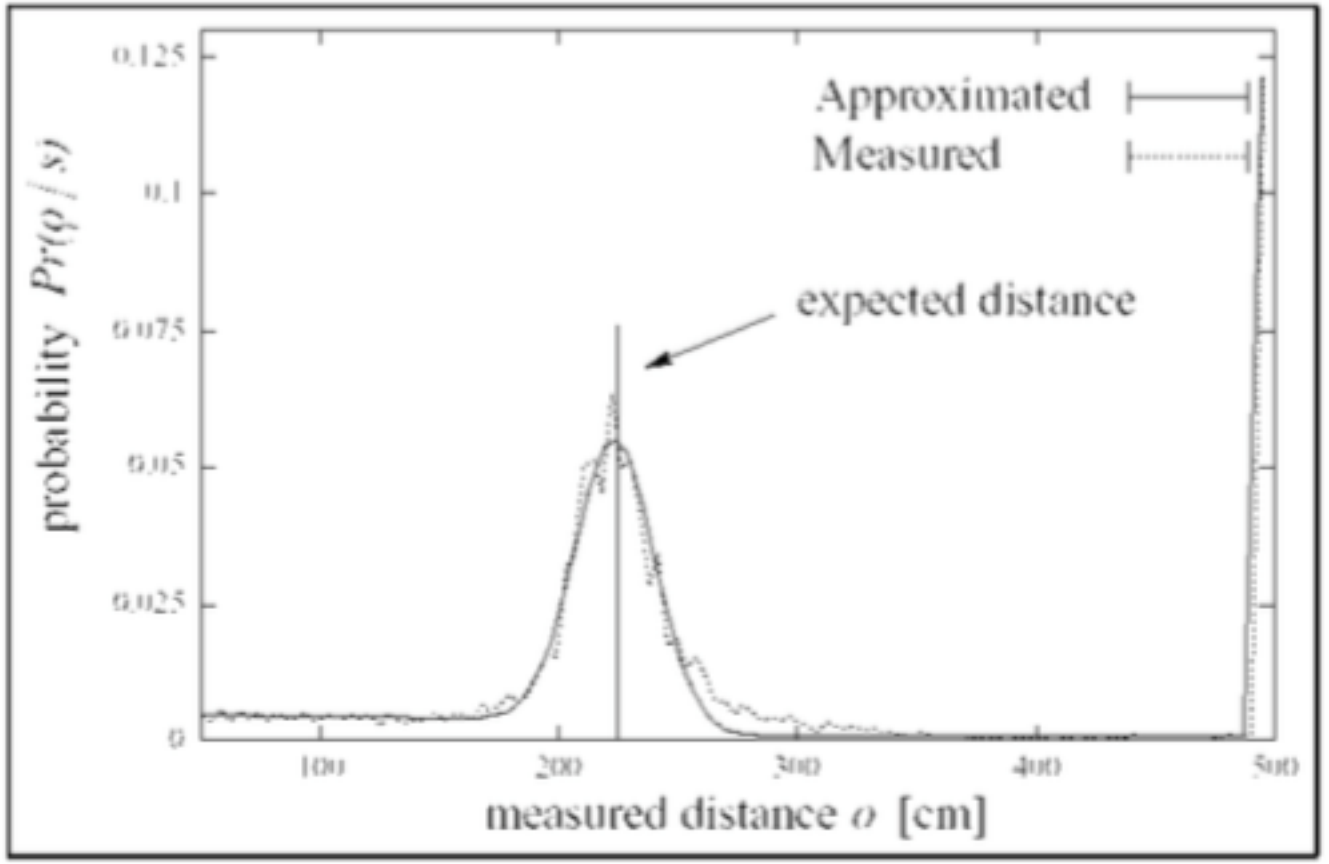
Given $o_{t-1}(m_i)$, x_t, z_t:

```
for all cells in m:
    if m_i is observed in z_t
        o_t(m_i) = o_{t-1}(m_i)+sensor_inverse(C_ij,z_t,x_t)
    else
        o_t(m_i) = o_{t-1}(m_i)
    endif
endfor
```

Sensor Model

$$p(z|m) = p(z = D|occ(i,j))$$

Probability of reading a range given known occupancy at known distance



Inverse Sensor Model

- If laser terminates at C_{ij} at distance D

$$\lambda(z = D|occ(i,j)) = \frac{p(z = D|occ(i,j))}{p(z = D|\neg occ(i,j))} \approx \frac{.06}{.005} = 12$$

$$\log_2 \lambda \approx +3.5$$

- If the laser passes through C_{ij}

$$\lambda(z > D|occ(i,j)) = \frac{p(z > D|occ(i,j))}{p(z > D|\neg occ(i,j))} \approx \frac{.45}{.9} = 0.5$$

$$\log_2 \lambda \approx -1.0$$

Implementation

- Rasterize each laser ray into the map to determine cells that are currently visible and free or occupied
- Convert known pose (x,y,θ) to start cell and reading (θ,d) to end cell in the map
- Use Breshenham's algorithm to update cells along the ray