Lecture 2
Robot description and state

Robotics: Design, Manufacturing, and Control **EE209AS – Fall 2016**
Prof. Ankur Mehta *Transcribed by: Tianrui Zhang*

2.1 Task Description of a Robot

The task description of a robot defines the goals of a robot, which includes sensing, decision making, and actuation. Often, the sensing and actuation goals of the description requires the robot to properly define the physical states of its own mechanical structure and any objects that it must interact with in the environment.

Example:

Move the paddle to the location of the ping-pong ball (see Fig. 2.1).

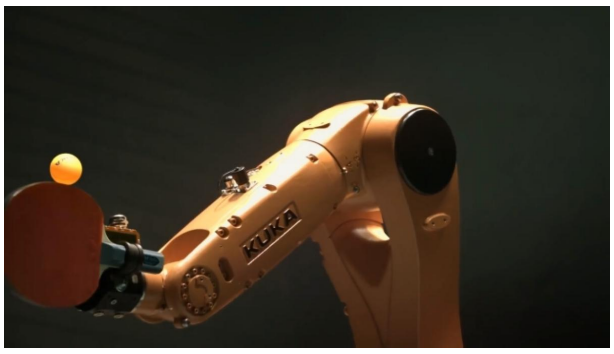


Figure 2.1: KUKA robot and Ping-Pong ball. *From [2]*

2.2 State description of a rigid body

The physical states of any rigid body is represented by its location and orientation/pose. For a rigid body that can be abstracted as a point mass, the description of its location is sufficient.

Examples:

1. The ping-pong ball can be viewed as a point mass, and its state is its location.
2. The state of the paddle contains both its location and orientation (e.g. If the paddle exists in 2D space, the orientation can be its rotation angle from a reference axis).

2.3 Mathematical Representation of the State

For the mathematical representation, we can borrow some concepts from linear algebra and geometry to help us describe the states.

- A **reference frame** contains an origin point, a set of orthogonal axes, and a unit length. These entities describe a set of physical reference points that uniquely defines a coordinate system and measurements within it.
- A **vector** is an ordered list of numbers that characterizes a point within a coordinate system. It has both a direction and a magnitude (length).

There may be a number of reference frames of interest. In particular, we will consider the following:

- **World / global reference frame**: A static, unchanging reference frame typically describing the environment that the rigid body exists in.
- **Body reference frame**: A reference frame with its origin and axes fixed relative to rigid body.

With these, we can define the state of the rigid body using the following values:

- **Position**: a vector describing the coordinates of the origin of body frame with respect to the world frame
- **Orientation**: a set of values describing the rotation of the body frame relative to the world frame. In 2D space, this can be the single angle, between corresponding axes in the body and world frames. In 3D space, this can be expressed in several different ways including: a rotation (or direction cosine) matrix, Euler angles, and a quaternion, which will be explained in Section 2.4 below.

Example:
Let o'_x, o'_y, o'_z denote the x, y, z coordinates of the origin of the body frame associated with the paddle in the coordinate system of the world frame (see Fig. 2.2). Then, the position vector $\boldsymbol{o'}$ of the paddle is given by:

$$\boldsymbol{o'} = \begin{bmatrix} o'_x \\ o'_y \\ o'_z \end{bmatrix}.$$

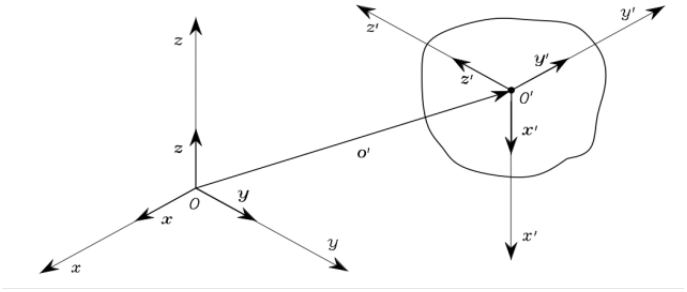


Figure 2.2: World and body reference frames, where the reference frame with origin O is the world frame and the reference frame with origin O' is the body frame. *From [1]*

2.4 3D rotations

2.4.1 Matrix Representation of Rotations

A matrix is a mathematical object that represents a linear transformation from one vector to another. We can represent the rotation of reference frames using a matrix $\boldsymbol{R}$:

$$\boldsymbol{R} = \begin{bmatrix} | & | & | \\ \boldsymbol{x'} & \boldsymbol{y'} & \boldsymbol{z'} \\ | & | & | \end{bmatrix} = \begin{bmatrix} x'_x & y'_x & z'_x \\ x'_y & y'_y & z'_y \\ x'_z & y'_z & z'_z \end{bmatrix} = \begin{bmatrix} \boldsymbol{x'^T x} & \boldsymbol{y'^T x} & \boldsymbol{z'^T x} \\ \boldsymbol{x'^T y} & \boldsymbol{y'^T y} & \boldsymbol{z'^T y} \\ \boldsymbol{x'^T z} & \boldsymbol{y'^T z} & \boldsymbol{z'^T z} \end{bmatrix},$$

where $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}$ are the unit vectors of original orthonormal reference frame and $\boldsymbol{x'}, \boldsymbol{y'}, \boldsymbol{z'}$ are the unit vectors of the rotated orthonormal reference frame (expressed using the coordinate system of the original frame).

Because the reference frames are orthonormal, $\mathbf{R}$ satisfies

$$\boldsymbol{x'^T y'} = 0 \quad \boldsymbol{y'^T z'} = 0 \quad \boldsymbol{z'^T x'} = 0 \quad \boldsymbol{x'^T x'} = 1 \quad \boldsymbol{y'^T y'} = 1 \quad \boldsymbol{z'^T z'} = 1.$$

Example:
A rotation matrix that rotates along the $\boldsymbol{z}$ axis by an angle α is as follows:

$$\boldsymbol{R_z(\alpha)} = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Because the rotation matrix usually contains trigonometric functions, it is also referred as **Direction Cosine Matrix(DCM)**.

2.4.2 Angular Representation of Rotations

Euler angles are angles to describe the orientation of a frame of reference via a sequence of simple rotations executed to achieve the final orientation. In three dimensional space, three angles are needed and each one of them is associated with a rotation about particular reference frame axis. There are two forms of Euler angles: classic Euler angles and Tait-Bryan angles.

Classic Euler angles: The first and third angles represent rotations about the same axis, while the second rotation is about one of the other two orthogonal axes. For example, ϕ can be the first rotation angle about the z-axis, θ the second rotation angle about the y-axis, and ψ be the third rotation angle about the z-axis again.

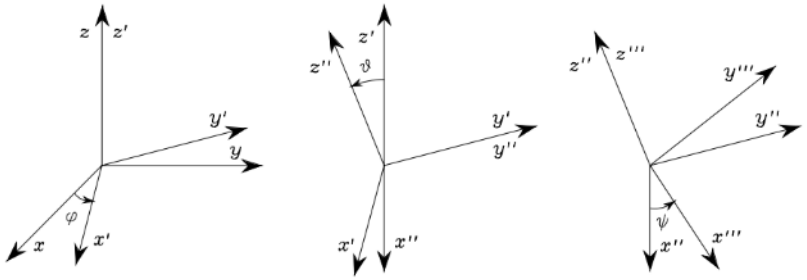


Figure 2.3: Classic Euler angles. *From [1]*

Tait Bryan angles: Tait Bryan angles are also known as yaw-pitch-row angles; each of the three rotations is about a unique axis. For example, ϕ can be the first rotation angle about the z-axis(roll), θ the second rotation angle about the y-axis(pitch), and ψ the third rotation angle along x-axis(yaw).

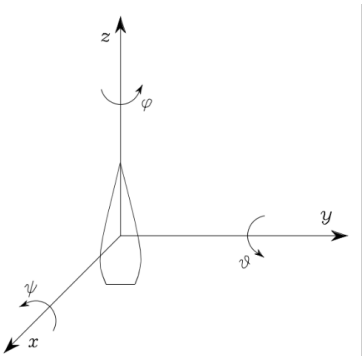


Figure 2.4: Tait Bryan angles. *From [1]*

For both conventions, the three elemental rotations may occur either about the axes of the original coordinate system, which remains motionless (extrinsic rotations), or about the axes of the rotating coordinate system, which changes its orientation after each elemental rotation (intrinsic rotations). Combined with choices for which axis sequence to use, this leads to total 12 possible axis choices for either of the Classic Euler or Tait-Bryan angle representations.

2.4.3 Axis-angle Representation of Rotations

Any orientation in space can be achived by a single rotation of the base frame about some vector in space. Thus, an orientation can be represented by the pair $\{\theta, \boldsymbol{r}\}$, with the unit vector $\boldsymbol{r}$ being the axis of rotation, and θ the angle of rotation about that axis.

2.4.4 Quaternion Representation of Rotations

Another way to represent the rotation is through a four-parameter representation $\boldsymbol{q} = \{q_a, q_i, q_j, q_k\}$ called a quaternion, where $q_a = \cos(\theta/2)$ and $\{q_i, q_j, q_k\} = \sin(\theta/2)\boldsymbol{r}$, with $\theta, \boldsymbol{r}$ as defined in the axis-angle representation above. The quaternion is normalized, so obeys:

$$q_a^2 + q_i^2 + q_j^2 + q_k^2 = 1$$

2.5 Transformations

We can describe the state of a rigid body within a world reference frame in three different, equivalent ways:

- we can give the values describing the position and orientation of the body frame in the world reference frame,
- we can describe a process by which a frame could rotate and translate from aligning with the world frame to aligning with the body frame, or
- we can provide the transformation that expresses a point in space using its coordinates in the world frame as a mathematical function of the coordinates of that same point in the body reference frame.

2.5.1 Linear tranformations

A point (vector) $\boldsymbol{p}$ is a geometric entity in space. It can be described by giving its coordinates in the world reference frame as the ordered set of numbers $\boldsymbol{p^0}$. Alternately, it can be described by giving its coordinates in a body reference frame as the vector (with potentially different values) $\boldsymbol{p^b}$. If the two frames are related by a linear transformation, the numerical values of these vectors are related by the corresponding transformation matrix $\boldsymbol{M_b^0}$ from frame b to frame o: $\boldsymbol{p^0} = \boldsymbol{M_b^0 p^b}$.

Examples of linear transformations include rotations, reflections, and projections, as well as stretching and skewing operations.

2.5.2 Rotations as linear transformations

Rotation is a linear transformation, with corresponding transformation matrix $\boldsymbol{M_b^0 = R_b^0}$ given by the DCM of frame b relative to frame 0.

2.5.3 Translations as affine transformations

Translations are not linear transformations, they are affine transformations. We cannot express this directly using a matrix multiplication.

2.5.4 Homogeneous Coordinates

For a frame b with both a rotational state $\boldsymbol{R}$ and translational offset $\boldsymbol{O}$, the coordinates of a vector $\boldsymbol{p}$ can be expressed using the following relation:

$$\boldsymbol{p^0 = O_b^0 + R_b^0 p^b},$$

where $\boldsymbol{p^b}$ and $\boldsymbol{p^0}$ are the coordinates of the vector expressed in the body and world frames respectively, $\boldsymbol{O_b^0}$ is the vector describing the origin of frame b in frame 0, and $\boldsymbol{R_b^0}$ is the rotation matrix of frame b in frame 0.

Note that the above equation can be rewritten in this way

$$\begin{bmatrix} \boldsymbol{p^0} \\ 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{R_b^0} & \boldsymbol{O_b^0} \\ \boldsymbol{0} & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{p^b} \\ 1 \end{bmatrix}.$$

The new vector formed by appending the number 1 as an additional element in the original vector is called the **homogeneous representation** of the original vector, which is usually denoted as $\widetilde{\boldsymbol{p}}$, and the matrix obtained is called the **homogeneous transformation matrix**, which is usually denoted as $\boldsymbol{T}$.

The use of homogeneous coordinates allows us to conveniently handle any general state of a rigid body as a linear transformation way such that

$$\widetilde{\boldsymbol{p}}^{\boldsymbol{0}} = \boldsymbol{T_b^0 \widetilde{p}^b}$$

.

2.6 Reference

1. Robotics: Modelling, Planning and Control 1st ed. 2009 Edition, Bruno Siciliano, Lorenzo Sciavicco, Luigi Villani, Giuseppe Oriolo

2. `http://assets.nydailynews.com/polopoly_fs/1.1717702.1394554418!/img/httpImage/image.jpg_gen/derivatives/article_750/pingpong12n-1-web.jpg`, 08/20/2016

Lecture 3
Forward and Inverse Kinematics

Robotics: Design, Manufacturing, and Control **EE209AS – Fall 2016**
Prof. Ankur Mehta *Transcribed by: Tianrui Zhang*

3.1 Robot Manipulator

Now, we can proceed to describe the physical dynamics of the robot itself.

Robot manipulator: a robot with fixed base, which consists of a sequence of rigid bodies (links) interconnected by means of articulations (joints) with a special part called end-effector that performs the task at the end of the link. There are two types of joints: revolute joint, which can rotate around the joint, and prismatic joint, which can move in a linear direction.

This mechanical structure is also referred as a kinematic chain.

Example:

An example robot manipulator(robotic arm).

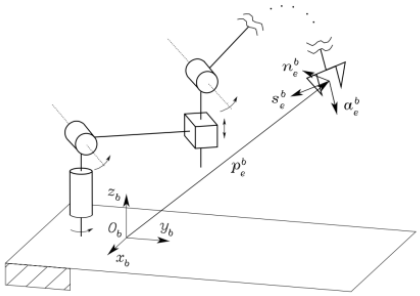


Figure 3.1: An example of robot manipulator. *From[1]*

3.2 State Variables and Degrees of Freedom

Before we proceed, we need the help of following concepts. **State variable** is any variable that helps to describe the physical state(position and pose) of an object. Sometimes, there are constraints imposed on the state variables.

Degrees of freedom(DOF) is used to describe the freedom of ways to change the state of a rigid body in three-dimensional space. DOF can be found by subtracting the number of constraints from the number of state variables.

If the number of state variables used to describe the state of a rigid body equals its DOF, then we have the minimal representation of the state.

Example:

1. Euler angle representation is a minimal representation of the orientation.

2. For a n link kinematic chain, the state of each link can be represented by at least six state variables(3 for position, 3 for pose using Euler angles). However, each link are constrained by a joint and can only move in one way. Therefore, the total degree of freedom is 6n-5n = n DOFs for this chain.

If the DOFs of robotic system <DOFs of our needs, we have under-actuated system.
If the DOFs of robotic system = DOFs of our needs, we have holonomic system.
If the DOFs of robotic system >DOFs of our needs, we have redundant system.

3.3 Joint space and Operational Space

Now we want describe the dynamics of our robotic system. In particular, we are more interested in defining the behavior of our end-effector mathematically.

Operational space: a set of state vectors in which states are represented by state variables of position and orientation. A typical operational space vector is

$$x_e = \begin{bmatrix} P_e \\ \phi_e \end{bmatrix}$$

, where p_e describes the end-effector position and s_e its orientation. Operational space vectors corresponds to the way of describing the state of rigid body we have talked so far. Yet another equivalent representation is joint space.

Joint space(Configuration space): a set of state vectors in which states are represented by joint variables. A typical joint space vector is

$$\boldsymbol{q} = \begin{bmatrix} q_1 \\ \vdots \\ q_n \end{bmatrix}$$

, where $q_i = \theta_i$ for a revolute joint and $qi = di$ for a prismatic joint.

We can then describe our end-effector in either operational space or joint space. The two spaces can be converted from one to another. Conversion from joint space to operational space is called forward kinematics, and conversion from operational space to joint space is called inverse kinematics, which will be explained in details later.

3.4 Modified Denavit-Hartenberg(DH) parameters

Forward kinematics is in general hard to do and doesn't give us the structure of robot.

Here, we will present a systematic way to do it. First, we need a systematic way to construct reference frame on each link. This method uses four parameters called **Denavit-Hartenberg parameters**.

The process for determining frame i is as follows:

1. Choose axis z_i along the axis of Joint i. Locate the origin O_i at the intersection of axis z_i with the common normal to axes z_{i+1} and z_i. Also, locate $O_i^{'}$ at the intersection of the common normal with axis z_{i+1}.

2. Choose axis x_i along the common normal to axes z_{i+1} and z_i with positive direction from Joint i to Joint i+1.

3. Choose axis y_i so as to complete a right-handed frame.

4. When two consecutive axes are parallel, the common normal between them is not uniquely defined.

5. When two consecutive axes intersect, the positive direction of x_i is arbitrary.

Then, the four parameters are defined as follows:

- $\boldsymbol{a_i}$ is distance between O_i and $O_i^{'}$,
- $\boldsymbol{d_i}$ is distance between $O_{i-1}^{'}$ and O_i,
- $\boldsymbol{\alpha_i}$ is angle between axes z_{i+1} and z_i about axis x_i to be taken positive when rotation is made counter-clockwise,
- $\boldsymbol{\theta_i}$ is angle between axes x_{i+1} and x_i about axis z_i to be taken positive when rotation is made counter-clockwise.

Example:

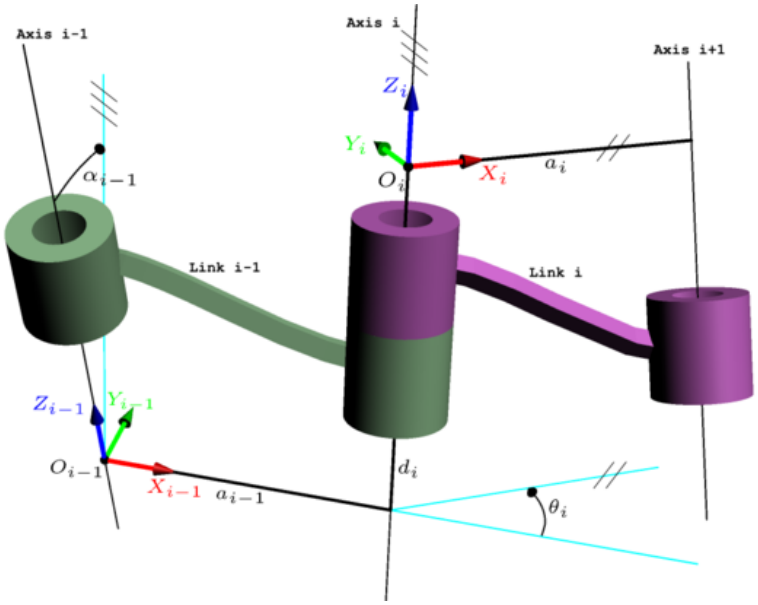


Figure 3.2: Modified D-H parameters *From[2]*

3.5 Forward Kinematics

Forward kinematics refers to a type of problems that uses the kinematic equations of a robot to compute the position of the end-effector with respect to the world reference frame from specified values for the joint parameters.

3.5.1 Forward Kinematics Transformation Matrix

Under the same principle, the overall transformation matrix that transfers vector in frame n to the world frame is

$$\boldsymbol{T_n^o = T_1^o T_2^1 T_3^2 ... T_n^{n-1}}$$

The forward kinematics problem is then simply to find such $\boldsymbol{T_n^o}$ and then calculate

$$\boldsymbol{\widetilde{p}^o = T_n^o \widetilde{p}^n}$$

3.5.2 Generalized Coordinate

In D-H parameter set up, we know that there are four parameters associated with each reference frame. Among all four parameters, α_i and a_i are always constant. Depending on the type of the joint, either d_i or θ_i will become variable.

In another words, the structure of the robotic linkage is uniquely defined by the combination of these variables.

Then, let's define

$$q_i = \epsilon_i d_i + (1 - \epsilon_i)\theta_i$$

, where ϵ_i equals one for prismatic joint and zero otherwise.

The **generalized coordinate** is then defined as

$$\boldsymbol{q} = \begin{bmatrix} q_1 \\ \vdots \\ q_n \end{bmatrix}$$

Note that the generalized coordinate is same as the joint space variable introduced earlier.

Since the structure of the robotic linkage is uniquely defined by the generalized coordinate, the transformation matrix is then a function of the generalized coordinate.

$$\boldsymbol{T_n^o} = f(\boldsymbol{q})$$

3.6 Inverse Kinematics

Inverse kinematics refers to a type of problems that uses the kinematics equations of a robot to determine the joint space variables that provide a desired position of the end-effector.

Inverse Kinematics is in general a hard problem. Here, we present a numerical way to solve it.

3.6.1 Differential Kinematics

Assuming the end-effector position is fixed relative to the frame n. Then, the end-effector position vector $\boldsymbol{x}$ in word frame is a function which purely depends on joint space variables.

$$\boldsymbol{x} = f(\boldsymbol{q})$$

At $\boldsymbol{q_0}$,

$$\boldsymbol{x_0} = f(\boldsymbol{q_0})$$

We have,

$$\delta\boldsymbol{x} = \boldsymbol{x} - \boldsymbol{x_0}$$

When we change $\boldsymbol{q}$ by $\boldsymbol{\delta q}$,

$$\delta\boldsymbol{x} = f(\boldsymbol{q} + \delta\boldsymbol{q}) - f(\boldsymbol{q})$$

As $\boldsymbol{\delta}$ goes to infinitely small, we have

$$d\boldsymbol{x} = f'(\boldsymbol{q})d\boldsymbol{q}$$

To find $f'(\boldsymbol{q})$, we know

$$\begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} f_1(\boldsymbol{q}) \\ \vdots \\ f_n(\boldsymbol{q}) \end{bmatrix}$$

Therefore,

$$\begin{aligned}
\partial x_1 &= \frac{\partial f_1}{\partial q_1}\partial q_1 + \frac{\partial f_1}{\partial q_2}\partial q_2 ... + \frac{\partial f_1}{\partial q_n}\partial q_n \\
\partial x_2 &= \frac{\partial f_2}{\partial q_1}\partial q_1 + \frac{\partial f_2}{\partial q_2}\partial q_2 ... + \frac{\partial f_2}{\partial q_n}\partial q_n \\
&... \\
\partial x_n &= \frac{\partial f_n}{\partial q_1}\partial q_1 + \frac{\partial f_n}{\partial q_2}\partial q_2 ... + \frac{\partial f_n}{\partial q_n}\partial q_n
\end{aligned}$$

In matrix form, we then have

$$d\boldsymbol{x} = \begin{bmatrix} \frac{\partial f_1}{\partial q_1} & \frac{\partial f_1}{\partial q_2} & \cdots & \frac{\partial f_1}{\partial q_n} \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ \frac{\partial f_n}{\partial q_n} & \frac{\partial f_n}{\partial q_2} & \cdots & \frac{\partial f_n}{\partial q_n} \end{bmatrix} d\boldsymbol{q}$$

Note that this matrix is actually a Jacobian matrix. Therefore, it is usually denoted as $J(\boldsymbol{q})$.
Therefore,

$$\begin{aligned} d\boldsymbol{x} &= f'(\boldsymbol{q})d\boldsymbol{q} \\ &= J_v(\boldsymbol{q})d\boldsymbol{q} \end{aligned}$$

So,

$$d\boldsymbol{q} = J_v^{-1}(\boldsymbol{q})d\boldsymbol{x}$$

Note that $J^{-1}(\boldsymbol{q})$ can be non-invertible. In such cases, we will use pseudo-inverse matrix instead.

3.6.2 Linear velocity and angular velocity

From above equations, we also have

$$\frac{d\boldsymbol{x}}{dt} = J_v(\boldsymbol{q})\frac{d\boldsymbol{q}}{dt}$$

This gives the **linear velocity** of the end-effector.

For **angular velocity** of the end-effector, we notice that only revolute joints contribute to the change of angular velocities and the contribution is $\boldsymbol{z_i}\dot{q}_i$. Therefore, we have

$$\begin{aligned} \boldsymbol{\omega} &= \sum_{i=1}^{n} \epsilon_i \boldsymbol{z_i}\dot{q}_i \\ &= \begin{bmatrix} \bar{\epsilon_1}\boldsymbol{z_1^o} & \bar{\epsilon_2}\boldsymbol{z_2^o} & ... \end{bmatrix} \dot{\boldsymbol{q}} \\ &= J_\omega \dot{\boldsymbol{q}} \end{aligned}$$

, where $\boldsymbol{z_1^o} = T_i^o \boldsymbol{z_i^i}$.

Therefore, we have

$$\begin{aligned} \begin{bmatrix} \boldsymbol{v} \\ \boldsymbol{\omega} \end{bmatrix} &= \begin{bmatrix} J_v(\boldsymbol{q}) \\ J_\omega(\boldsymbol{q}) \end{bmatrix} \dot{\boldsymbol{q}} \\ &= J(\boldsymbol{q})\dot{\boldsymbol{q}} \end{aligned}$$

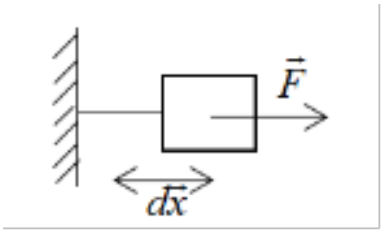
3.6.3 Inverse Kinematic Algorithm

1. Given an operational space vector $\boldsymbol{x}_0$ and corresponding joint space vector $\boldsymbol{q}_0$ as initial point
2. Pick a step size $\delta\boldsymbol{x}$ towards goal
3. Find $\delta\boldsymbol{q} = J^{-1}(\boldsymbol{q}_i)\delta\boldsymbol{x}$ and calculate $\boldsymbol{q}_{i+1} = \boldsymbol{q}_i + \delta\boldsymbol{q}$
4. Calculate corresponding operational space vector $\boldsymbol{x}_{i+1}$ using forward kinematics
5. Check if $\boldsymbol{x}_{i+1}$ is close enough to the destination point. If yes, you have found $\boldsymbol{q}$. If not, go back to step 2 with new $\boldsymbol{x}$ and $\boldsymbol{q}$

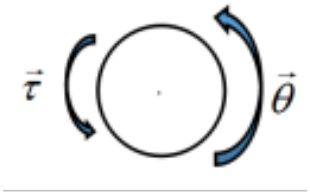
3.7 Static Forces

Static forces analysis is applied on the robotic arm to find out relationship between the forces applied to the end-effector and forces applied to the joints at the static equilibrium states.

The total work done by a single 1-DOF prismatic joint is $dw = \boldsymbol{F} \cdot d\boldsymbol{x}$, where dw is the total amount of work done by the joint, $\boldsymbol{F}$ is the force vector applied on the joint and $d\boldsymbol{x}$ is the distance vector of the joint, as shown in the picture below:



The total work done by a single 1-DOF revolute joint is $dw = \boldsymbol{\tau} \cdot d\boldsymbol{\theta}$, where dw is the total amount of work done by the joint, $\boldsymbol{\tau}$ is the torque vector applied on the joint and $d\boldsymbol{\theta}$ is the rotation vector of the joint, as shown in the picture below:



At joint space, if we can put all force and torque vectors into a single vector,

$$\boldsymbol{\tau_F} = \begin{bmatrix} \boldsymbol{F_1} \\ \boldsymbol{\tau_2} \\ \vdots \\ \boldsymbol{F_n} \end{bmatrix}$$

, then the total work done by the robotic arm is

$$\begin{aligned} dw_q &= \boldsymbol{F_1} \cdot d\boldsymbol{x_1} + \boldsymbol{\tau_2} \cdot d\boldsymbol{\theta_2} + ... + \boldsymbol{F_n} \cdot d\boldsymbol{x_n} \\ &= \boldsymbol{\tau}_F \cdot d\boldsymbol{q} \\ &= \boldsymbol{\tau}_F^T d\boldsymbol{q} \end{aligned}$$

, where $d\boldsymbol{q}$ is the joint space variable vector.

At operational space, the total work done onto the end-effector consists of the linear part and the angular part. Therefore, the total work done is

$$\begin{aligned} dw_x &= \boldsymbol{F} \cdot d\boldsymbol{s} + \boldsymbol{\tau} \cdot d\boldsymbol{\theta} \\ &= \boldsymbol{F} \cdot \boldsymbol{v} dt + \boldsymbol{\tau} \cdot \boldsymbol{\omega} dt \\ &= (\boldsymbol{F}^T \boldsymbol{v} + \boldsymbol{\tau}^T \boldsymbol{\omega}) dt \\ &= \begin{bmatrix} \boldsymbol{F} \\ \boldsymbol{\tau} \end{bmatrix}^T \begin{bmatrix} \boldsymbol{v} \\ \boldsymbol{\omega} \end{bmatrix} dt \end{aligned}$$

From above, we have $d\boldsymbol{x} = J_v(\boldsymbol{q}) d\boldsymbol{q}$. Therefore,

$$\begin{aligned} d\boldsymbol{x}/dt &= J_v(\boldsymbol{q}) d\boldsymbol{q}/dt \\ \boldsymbol{v} &= J_v(\boldsymbol{q}) d\boldsymbol{q}/dt \\ \boldsymbol{v} \cdot dt &= J_v(q) d\boldsymbol{q} \end{aligned}$$

Similarly, we have $\boldsymbol{\omega} \cdot dt = J_\omega(q)d\boldsymbol{q}$, and

$$\begin{aligned}\begin{bmatrix} \boldsymbol{v} \\ \boldsymbol{\omega} \end{bmatrix} dt &= \begin{bmatrix} J_v(\boldsymbol{q}) \\ J_w(\boldsymbol{q}) \end{bmatrix} d\boldsymbol{q} \\ &= J(\boldsymbol{q})dq\end{aligned}$$

By conservation of energy, we have

$$\begin{aligned}\boldsymbol{\tau}_F^T d\boldsymbol{q} &= \begin{bmatrix} \boldsymbol{F} \\ \boldsymbol{\tau} \end{bmatrix}^T J(\boldsymbol{q})d\boldsymbol{q} \\ \boldsymbol{\tau}_F^T &= \boldsymbol{F}_g J \\ \boldsymbol{\tau}_F &= J^T \boldsymbol{F}_g\end{aligned}$$

, where F_g is called the **generalized force vector**.

3.8 Reference

1. Robotics: Modelling, Planning and Control 1st ed. 2009 Edition, Bruno Siciliano, Lorenzo Sciavicco, Luigi Villani, Giuseppe Oriolo
2. `https://en.wikipedia.org/wiki/Denavit%E2%80%93Hartenberg_parameters`, 08/20/2016
3. `https://www.youtube.com/watch?v=rA9tm0gTln8`, 08/20/2016
4. `http://assets.nydailynews.com/polopoly_fs/1.1717702.1394554418!/img/httpImage/image.jpg_gen/derivatives/article_750/pingpong12n-1-web.jpg`, 08/20/2016

Lecture 5
End effectors

Robotics: Design, Manufacturing, and Control **EE209AS – Fall 2016**
Prof. Ankur Mehta *Transcribed by: Tianrui Zhang*

5.1 End effectors

End effector is the device at the end of a kinematic chain that can effect (impact or influence) on the environment. Its biological analogy is any body part that can do "physical work" on the environment, such as tongue, fingers, and hands.

The robotic end effectors are designed to be well matched to robot's task, or at least general enough to cover it. Therefore, they are usually special cases of a general effector.

These effectors simply work by receiving commands from controllers to produce desired effect on the environment based on robot's task.

In general, there are three different types of end effectors:

1. custom passive device: one example is ping pong paddle
2. custom active device: one example is the extruder of 3D printer
3. general device: one example is human hands

It is worth mentioning that end effector can also be used for locomotions. Wheels, legs, and propellers are also considered as end effectors.

5.1.1 Goals of end effectors

The goal of a end effector describes its ideal kinematic states and is inextricably linked with its task definitions.

- Full 6-DOF position and orientation
- Reduced DOF subspace

  The end effector may only have freedom in its position or orientation. It may also only stay in a particular region near or out of a particular region from the target object.
- Force specification only

  Sometime the end effector may only need to satisfy certain force requirements, such as generating enough force to support an object.
- Combination of both

  This is commonly seen in the case of human-robot interaction scenarios due to safety concerns.

To put end effector into its goal states, we can translate goals into equations in operational space coordinates, then use our knowledge in inverse kinematics to determine equations in joint space. However, things can get complicated quickly due to multiple chains, loops, etc.

5.1.2 Actuators

To specifically generate motion that end effectors need, we need actuators. **Actuators** are one class of **transducer**, which is the device that converts one form of energy to another. The energy can be in form of chemical, mechanical, thermal, optical, etc.

There are two types of actuators: **active actuators** and **passive actuators**.Examples of active actuators are engines, motors, and solenoids. Examples of passive actuators are springs and water wheels.

Electric Motors

Electric motor is the most common actuator in robotics. It transduces electrical energy to rotational mechanical energy. The principle behind standard electric motors is Lorentz force, the force between current in a wire and the magnetic field. Usually, motor consists of two main parts: the **rotor**, which turns the shaft to deliver the mechanical power, and **stator**, which is the stationary part of a rotary system.

There are many types of electromagnetic motors: AC vs DC, brushed vs brushless. Here, we focus on **DC brushed motor**, which is also the most common motor.

DC brushed motor

1. **Structure**

   The structure of the DC brushed motor is shown below:

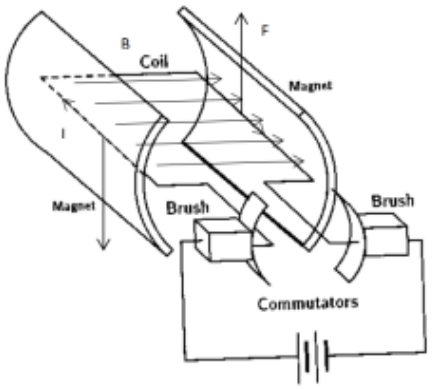


Figure 5.1: DC motor. *From[2]*

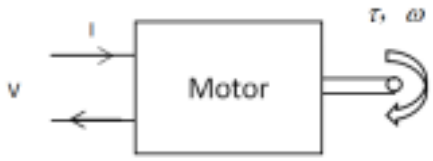
   Here, the stator is the permanent magnet and the rotor is the DC coil.

2. **Principles**

   According to Lorentz force, the force exerted on wire conducting a current in a magnetic field is $\vec{F} = \vec{i} \times \vec{B}$, where F is force vector, i is the current vector and B is the magnetic field flux density vector. The directions of F vectors are shown above given the directions of i and B.

   The brushes are needed there to keep the motor rotating. Otherwise, the coil will stay stationary once it reaches equilibrium position.

3. **Power and Efficiency**



From the input side, the motor is an electric device and the power $P_E = V * I$.
From the output side, the motor is an mechanical device and the power $P_M = \tau * \omega$.

The efficiency of the motor is, therefore, $E = P_M/P_E$. It can be up to 90% and down to 50%. The rest of energy is turned into resistive heat in wires or heat from mechanical friction losses. E depends on a lot of things and is not a constant in most cases.

4. **Relationship betwee τ and ω**

   The angular velocity and the torque have an inverse relationship. When the coil moves inside the magnetic field, the motion of wires cutting magnetic field lines generates induced voltage that has opposite polarity of applied voltage. When ω increases, the induced voltage will also increase. The current in the running motor $i = (V_{applied} - V_{induced})/R_{ckt}$ will therefore drop. This will in turn decrease the force generated due to equation $\vec{F} = \vec{i} \times \vec{B}$ and hence torque. The reverse is also true.

   In general the inverse relation can be well approximated by a linear model and is shown below. We can also plot the power curve in the same figure.

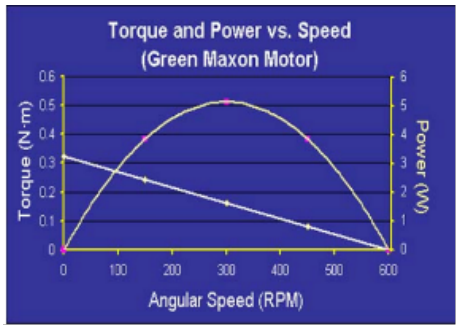


Figure 5.2: Torque and power curve of motor. *From[3]*

The angular velocity when load is zero is called **no load speed** and the torque when angular velocity is zero is called **stall torque**. The peak power happens when both ω and τ are half of their maximum values.

5. **Controls**

   - To control the speed of the motor, we can use a technique called **Pulse-width modulation (PWM)**. We can control the power supplied to electrical devices by varying the portion of "on" and "off" time(Duty Cycle) of an electric pulse during one period. Example of PWM waveforms are shown below:

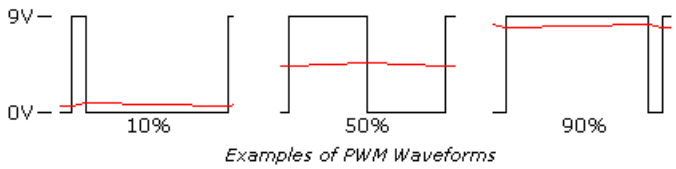


Figure 5.3: PWM wave. *From[4]*

One advantage of using PWM is that we can achieve similar effect of varying voltage levels by simply changing the duty cycle. For example, 9V with 10% duty cycle has similar effects as 0.9V supply.

- To control the direction of the motor, we can use a circuit called **H-bridge**. H-bridge is simply a circuit design shown below:

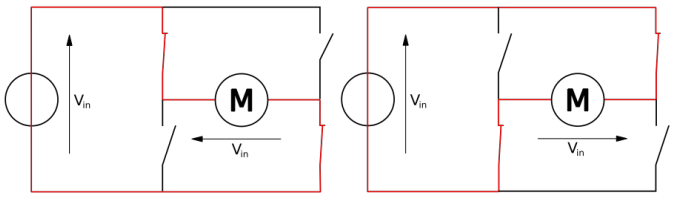


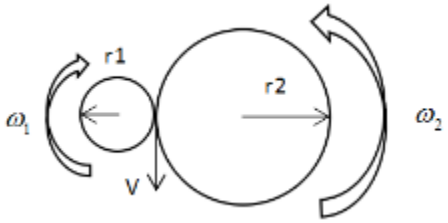
Figure 5.4: H-bridge. *From[5]*

By open and close certain switches in the circuit, we can reverse the rotational direction of the motor, as shown in the picture. In practise, we usually use MOSFETs with diodes instead of mechanical switches.

- To break the motor, an easy way is to simply connect the two terminals of the motor while power is applied. The induced back EMF will help to break the motor when applied voltage is zero in this case.

Transmission

Sometimes we demand low speeds and high torques during actuating. However, motors tend to have high speeds with low torques in optimal operating conditions.Therefore, we need transmission gears to transfer to the mechanical power. If we have a two gear system as following:



Because the magnitudes of linear velocity and force exerted at touching point are same for both gears,

we have:

$$\begin{aligned} v &= r_1\omega_1 \\ &= r_2\omega_2 \\ \omega_2 &= \omega_1 r_1/r_2 \\ F &= \tau_1/r_1 \\ &= \tau_2/r_2 \\ \tau_2 &= \tau_1 r_2/r_1 \end{aligned}$$

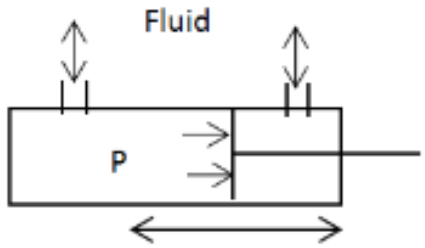
Here, $r_2/r_1 = N_2/N_1$, where N is number of teeth on the gear and it is proportional to the radius of the gear, is called the **gear(velocity) ratio** of the gear system.

There will be some friction losses during transmission. So the actual torque will be $\tau_2 = \epsilon\tau_1 r_2/r_1$ and the power transmitted is $P_1 = \tau_1\omega_1 = \tau_2\omega_2/\epsilon$

Other Actuators

Motor is a revolute actuator, but sometimes we need prismatic actuators as well. This can be done by adding mechanical structure to motors to convert revolute motions to prismatic motions.

This can also be done by using linear pressure actuators, as shown below. In general, there are two types of pressure actuators that are commonly used: hydraulics and pneumatics. Hydraulics actuators are incompressible and closed, and pneumatics actuators are compressible and open.



$$\vec{F} = \vec{P}\dot{A}$$

5.2 Reference

1. Robotics: Modelling, Planning and Control 1st ed. 2009 Edition, Bruno Siciliano, Lorenzo Sciavicco, Luigi Villani, Giuseppe Oriolo
2. `http://mechatronics.mech.northwestern.edu/design_ref/actuators/motor_theory.html`
3. `http://lancet.mit.edu/motors/motors3.html`
4. `http://www.allaboutcircuits.com/textbook/semiconductors/chpt-11/pulse-width-modulation/`
5. `https://en.wikipedia.org/wiki/H_bridge`

Lecture 6
Sensors

Robotics: Design, Manufacturing, and Control **EE209AS – Fall 2016**
Prof. Ankur Mehta *Transcribed by: Tianrui Zhang*

6.1 Sensors

Besides actuators, sensors are of great importance in robotic system as well. In essence, a sensor is an transducer to detect energy changes in its environment and output corresponding signals, typically electrical signals. It can output other kinds of signals, but we will only consider electrical signals here.

6.1.1 Goals of sensors

The goal of using sensors in a robotic system is simply to determine states, including states of the environment the robot exists in(**Environment States**) and states of the robot itself(**Robot States**).

For example, sensors can be used to measure the distance of obstacles in the environment or to measure the torques generated at joints to determine if a robotic arm is operating normally.

6.1.2 Types of sensors

There are many ways to differentiate different types of sensors:

1. **Proprioceptive vs. Exteroceptive**

   Proprioceptive sensors are used to measure the internal states of a robot. An example is the GPS sensor.
   Exteroceptive sensors are used to measure the external states of a robot. An example is the vision camera for computer vision sensing.

2. **Active vs. Passive**

   Active sensors actively produce energy to do the measurement. An example is the IR proximity sensor.
   Passive sensors passively consume externally generated energy. An example is the thermal sensors.

3. **Digital vs. Analog**

   Digital sensors output digital signals, whereas analog sensors output analog signals.

Example:

Figure 6.1: A rotary encoder. *From[2]*

A rotary encoder is a sensor to measure angular motions. It turns the mechanical energy from angular motions to electrical signals. A rotary encoder installed in the revolute joint inside a robotic arm is therefore a proprioceptive and passive sensor.
Depending on the design of the encoder, it can be as simple as a potentiometer and therefore an analog sensor. It can also output motion information in a binary fashion by setting its several output pins to different "high" or "low" values.

6.1.3 Information processing

For most of the time, sensor measurements will contain different kinds of noises and disturbances from the environment. This makes these measurements somewhat unreliable.
Besides the effects from noises and disturbances, many times, we want to combine sensory data from disparate sources as well. This is known as sensor fusion.
Therefore, we need to process these data before using them. Here, we present a signal processing algorithm that is widely used in the robotic field, which is **Kalman filtering**.

Kalman filter

Kalman filtering is a linear quadratic estimation algorithm that uses both measurements observed over time and theoretical predictions of target variables to produce best estimates of those variables.

Intuitively, it works by adjusting the weight, called **Kalman gain**, between the theoretical prediction and the actual measurements from the sensor to gain a more accurate prediction. The algorithm will also adjust the Kalman gain automatically based on the observations.

A Kalman filter is an optimal estimator in the sense that, if all noise is Gaussian, then it minimises the mean square error of the estimated parameters.

Example:
The general algorithm and a simple calculation example can be found at link from reference 3.

Extended Kalman filter

When the system is not a linear system, we can use extended Kalman filter algorithm to find the best estimate. Most of the algorithm stay the same as Kalman filter, but we need to linearize the state transition function and the observation function at calculation point.

Example:
The algorithm can be found at link from reference 4.

6.2 Reference

1. Robotics: Modelling, Planning and Control 1st ed. 2009 Edition, Bruno Siciliano, Lorenzo Sciavicco, Luigi Villani, Giuseppe Oriolo

2. `https://www.sparkfun.com/products/10790`, 09.20.2016

3. `http://bilgin.esme.org/BitsAndBytes/KalmanFilterforDummies`, 09.20.2016

4. `https://en.wikipedia.org/wiki/Extended_Kalman_filter`, 09.21.2016

Lecture 7
Controller

Robotics: Design, Manufacturing, and Control **EE209AS – Fall 2016**
Prof. Ankur Mehta *Transcribed by: Tianrui Zhang*

7.1 Controller

In control theory, there are some basic concepts:

- **Plant** is the subsystem to be controlled
- **Sensor** is the part in the system that measures the quantity to be controlled and feeds it back to the controller
- **Actuator** is the part that is controlled by controller and can affect the system
- **Controller** is the part which manages the actuator to make the Plant's behaviours same as our goal.
- **Control algorithm** is the algorithm used by the controller and usually can be represented by a transfer function.

Control systems are usually represented as block diagrams. The plant model is therefore as follows:

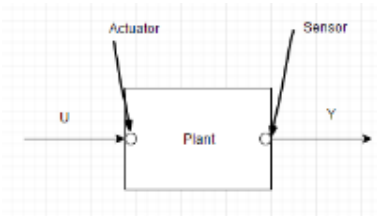


Figure 7.1: Block diagram model for plant

U is the input to the plant and Y is the output from the plant.
There are two types of controller: open-loop controller and close-loop controller.

7.1.1 Open-loop controller

Open-loop control is also known as feedforward control. In open-loop control systems, there is no feed back to the controller. So, the output is generated purely based on the input.
The block diagram for open-loop control is shown below,

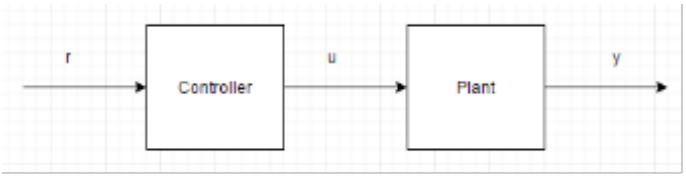


Figure 7.2: Block diagram model for open-loop control

, where u and y in the figure are same as before. r is the goal of the controller. Therefore, the controller needs to drive y->r by setting u to the plant based on its control model.

Mathematically, if we let C be the transfer function of the controller and H be the transfer function of the plant, then the final transfer function from r to y is HC and the goal of the controller simply becomes setting C such that the magnitude of final transfer function is near 1 given H.

However, this is generally hard for open-loop control, because there are usually unknown disturbances and noises from many aspects and they are hard to be fully captured by a static control model.

7.1.2 Close-loop controller

In a close-loop control system, the current output is fed back to the controller and the control model then takes into account the error between the measured output and the goal.
The general block diagram for close-loop control is shown below,

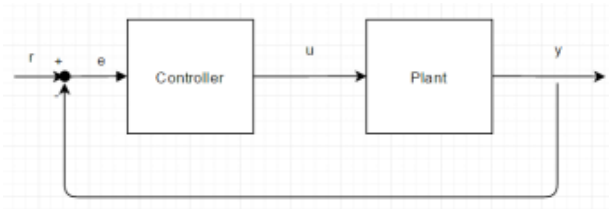


Figure 7.3: Block diagram model for closed-loop control

, where r, u, y are same as before and e = r-y.

Let C be the transfer function of the controller and H be the transfer function of the plant, then the final transfer function from r to y is $T = \frac{Y}{R} = \frac{HC}{1+HC}$. Similarly, the goal of the controller is to set C such that the magnitude of final transfer function is near 1 given H. However, the feedback process may take time. Therefore, the final transfer function only need to stabilized to one with long enough time.

There are many types of close-loop controllers, and among all these controllers, bang-bang controllers and PID controllers are by far the most popular ones.

Bang-bang controller

A bang-bang controller is simply a controller that turns on when r¡y and turns off when r¿y. Bang-bang control works but it will create oscillations around the goal and is not very efficient.

PID controller

PID is the abbreviation for **Proportional-Integral-Derivative**. It therefore contains three components: the proportional part, the integral part, and the derivative part. It's worth mentioning that PID controller does not have to contain all part at the same time, it may contain any combination of the three parts.

1. **P control**

   The proportional control produces an output value that is proportional to the current error value. The formula is

   $$u(t) = K_p e(t)$$

Intuitively, this works because bigger error is translated to higher input to plant, which drives the output faster to its desired value. When the error is small, the input will also be small.

However, the P control may have a **steady-state error**. That is, the error term will be nonzero when system becomes stable. This can be seen by applying final value theorem to error term.

$$e(\infty) = \lim_{s \to 0} R(s)(1 - T(s))$$

It may not go to zero.

2. **PI control**

   The added integral term is proportional to both the magnitude of the error and the duration of the error. The formula is

   $$u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau$$

   Intuitively, it provides a faster rate of convergence by adding another term and solves the steady state error because, if the steady-state error exists, it will drive the PI controller to generate input which will reduce itself.

3. **PID control**

   The full PID controller adds the integral term, which is proportional to the slope of the error over time.

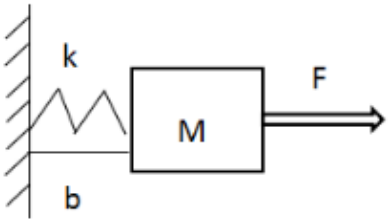
   $$u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{de(t)}{dt}$$

   Intuitively, the added term helps to predict system behavior and thus improves stability of the system.

It's worth mentioning that the choice of K_p, K_i, and K_d has a great impact on the system behaviour. They may even unstablize the system when chosen carelessly. Therefore, we need to find suitable values for these constants. This process is known as **tuning**.

Example:

Suppose we have a damping spring mass problem shown in the figure.



The physical model gives us:

$$m\ddot{x} + b\dot{x} + kx = F$$

Let m=1, b=10, k=20, and F=1, and let F be the input of the system and x be the output of the system. The transfer function H of the plant is then $H = \frac{X}{F} = \frac{1}{s^2+10s+20}$

When using a PID controller, the overall transfer function $T = \frac{X}{F_r} = \frac{K_d s^2 + K_p s + K_i}{s^3 + (10+K_d)s^2 + (20+K_p)s + K_i}$

After trial and error, we can set $K_p = 350, K_i = 300$, and $K_d = 50$ and the system will be a stable system.

7.2 Reference

1. `https://en.wikipedia.org/wiki/PID_controller`, 09/23/2016
2. `http://ctms.engin.umich.edu/CTMS/index.php?example=Introduction&section=ControlPID`, 09/23/2016

Lecture 8
Motion Planning

Robotics: Design, Manufacturing, and Control **EE209AS – Fall 2016**
Prof. Ankur Mehta *Transcribed by: Tianrui Zhang*

8.1 Path Planning

The goal of **path planning** is simply to get the robot from a starting point to a goal destination with a high chance of the presence of obstacles.

Depending on if the information about obstacles are available, the motion planning algorithms can be divided into online algorithms, in which the information is unavailable, and offline algorithms, in which the information is available.

8.1.1 Online Algorithm

Assuming that we only know the location of the goal, the state of our robot and the state history, then we can use **bug algorithm** to get to the goal.

Bug Algorithm

There are many variations of the bug algorithm. A common one is as follows.

1. Head toward goal on the line connecting the starting point and the goal
2. When encountering obstacles, move along obstacles and remember the path until you hit the line again closer to the goal.
3. Continue moving to the goal

An example is illustrated below.

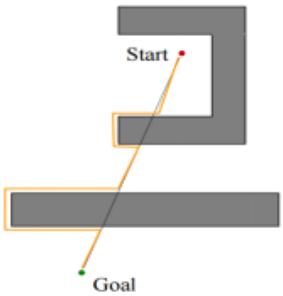


Figure 8.1: Bug algorithm. *From[2]*

8.1.2 Offline algorithm

The general offline path planning problem is defined as: given robot in configuration space C, set of obstacles in configuration space C_{obs}, the initial configuration q_i and the goal configuration q_f, find a path function x that maps values from [0,1] to configuration space such that $x(0) = q_i, x(1) = q_f$, and x(s) $\notin C_{obs} \forall s$.

Here, we introduce four popular offline algorithms:

1. **Exact Decomposition**

   The idea behind exact decomposition is that the free space of the environment can be decomposed into smaller regions, and then a path between the starting point and the goal can be constructed on these regions. A typical choice for these small regions are convex polygons. After this decomposition, we can treat each region as a single point and then draw an edge between any two points that represent adjacent regions in decomposed spaces.

   By doing so, we just turned the path planning problem into a graph search problem. We can then use algorithms we know in graph search to solve the problem. For example, we can use **BFS, DFS, Dikstra and A*** algorithms.

   An example is illustrated below.

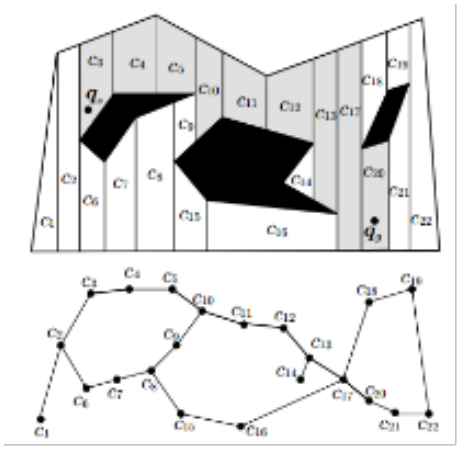


Figure 8.2: Exact decomposition. *From[1]*

2. **Discretization**

   The idea behind discretization is exactly same as exact decomposition. The only difference is that, instead of decompose the free space into random polygons, we divide them into regular sized squares. Then, we can use the same graph search algorithms to solve the problem.

   An example is illustrated below.

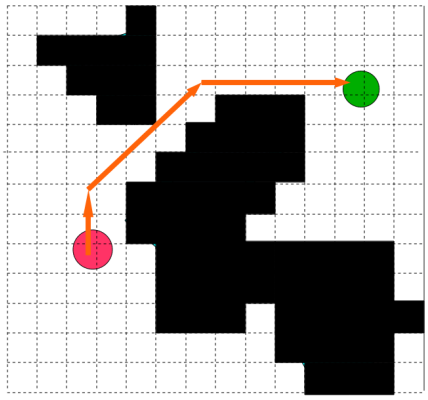


Figure 8.3: Discretization. *From[3]*

3. **Artificial Potential Fields**

   The idea behind the potential field algorithm is that we can make an analogy between the motion of robot in the environment and the motion of a particle under the influence of a potential field. If we give the starting point high potential and the destination point low potential, then the robot will be naturally "driven" by the filed and "run "to the goal position. At the mean time, we just need to assign really high potentials to where obstacles sit. The robot will then automatically avoid those obstacles.

   After creating the artificial potential fields, we just turned the path planning problem into an optimization problem. We can then use algorithms we know in optimization to solve the problem. For example, we can use **gradient descent method or Newton's method**.

   An advantage of this algorithm is that it is usually very efficient because no computational power is wasted to calculate things that has no direct relevance to the robot's current motion. However, the biggest drawback is that the algorithm may fall into a local minima and thus never reach the goal position.

   An example is illustrated below.

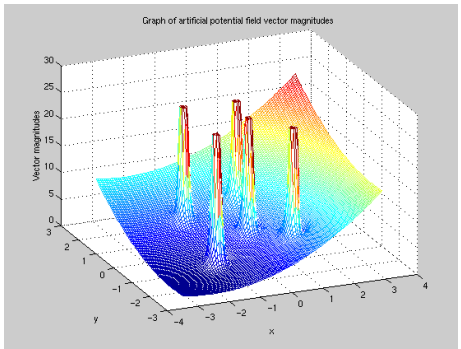


Figure 8.4: Artificial potential fields. *From[4]*

4. **Probabilistic Roadmap**

   The idea behind probabilistic roadmap algorithm is to randomly sample points in free space and construct a connectivity graph based on these points until the goal point is included.

   The algorithm consists of two phases:
   In the construction phase, first, a random point in free configuration space is selected.It is then checked against C_{obs}. If a conflict exists, the point is discarded. Otherwise, if the point meets certain distance

requirements, for instance, it has at least k nearest neighbors or its neighbors are less than a predetermined maximum distance d, it will be added to the connectivity graph. The process repeats until the goal point is in the graph.
In the search phase, the problem is again turned into a graph search problem, and we can then use algorithms mentioned above to solve the problem.

An example is illustrated below.

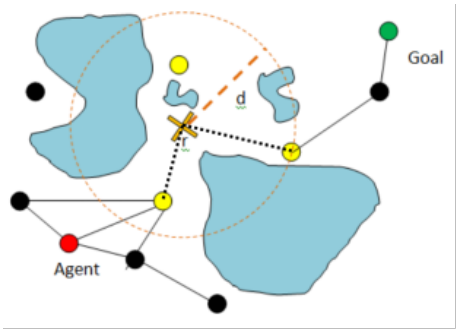


Figure 8.5: Probablistic roadmap. *From[3]*

An algorithm is **complete** if it terminates in finite time and returns a solution if one exists or declares failure if one does not. This is in general hard, and all known complete algorithms have complexity exponential in DOFs. However, if we relax some guarantees, the algorithms may not still be complete, but will be more efficient to implement.

Below is a table summarizing the completeness and complexity of all four algorithms mentioned:

Algorithms	Completeness	Complexity
Exact Decomposition	complete	exponential
Discretization	reselution complete	exponential
Artificial Potential Fields	not complete	polynomial
Probabilistic Roadmap	probabilistic complete	???

8.2 Trajectory planning

The above path planning problem does not consider the dynamics of the robot. When taking that into account, the problem becomes a trajectory planning problem.

The general trajectory planning problem is defined as: given robot in configuration space C, set of obstacles in configuration space C_{obs}, the initial configuration q_i, the goal configuration q_f, and equations describing the robot dynamics $\dot{c} = f(c, u)$, where u is the system input, find a path function x that maps values from [0,1] to configuration space such that $x(0) = q_i, x(1) = q_f$, and x(s) $\notin C_{obs} \forall s$.

To solve trajectory planning problem, one can use an algorithm called **Rapidly-exploring random tree**.

8.2.1 Rapidly-exploring random tree

The idea behind RRT algorithm is very similar to probabilistic roadmap algorithm. It also involves randomly selecting a point in free configuration space. However, every time a point is selected, it is checked against the dynamic equations we have to see if it is reachable from current spanning tree. If it is not, we simply discard it and go to next iteration. In RRT, points in unexplored space usually have higher chances to be chosen next time.

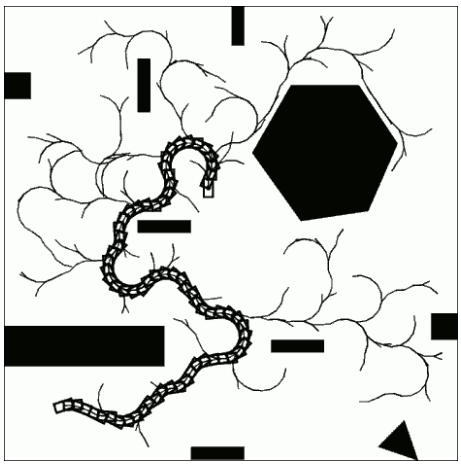


Figure 8.6: RRT. *From[4]*

8.3 Reference

1. Robotics: Modelling, Planning and Control 1st ed. 2009 Edition, Bruno Siciliano, Lorenzo Sciavicco, Luigi Villani, Giuseppe Oriolo
2. `https://www.cs.cmu.edu/~motionplanning/lecture/Chap2-Bug-Alg_howie.pdf`, 09/24/2016
3. `http://www.gamasutra.com/blogs/MattKlingensmith/20130907/199787/Overview_of_Motion_Planning.php`, 09/24/2016
4. `http://people.csail.mit.edu/lpk/mars/temizer_2001/Method_Comparison/Images/demo_vector_field_1.gif`, 09/24/2016
5. `http://aigamedev.com/static/theory/rrt-nonholonomic.png`, 10/19/2016

Lecture 9
Mechanics

Robotics: Design, Manufacturing, and Control **EE209AS – Fall 2016**
Prof. Ankur Mehta *Transcribed by: Tianrui Zhang*

9.1 Review of basic concepts

9.1.1 Forces

Force is the cause of linear motions. Mathematically, it can be represented by a vector quantity with both magnitude and direction.

9.1.2 Torques

Torque is the cause of angular motions, and it is generated by offset forces. Mathematically, it is also represented as a vector quantity. If torque is expressed about rotation center o, we have

$$\vec{\tau}_o = \vec{r}_{oF} \times \vec{F}$$

If torque is expressed at a random point p, we have

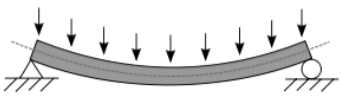
$$\begin{aligned}\vec{\tau}_p &= \vec{r}_{pF} \times \vec{F} \\ &= \vec{\tau}_o + \vec{r}_{op} \times \vec{F}\end{aligned}$$

with

$$\vec{r}_{pf} = \vec{r}_{oF} - \vec{r}_{op}$$

9.1.3 Distributed loads

Distributed loads is a distribution of load forces on the object instead of forces acting on a single touching point. The forces are usually expressed as force per unit(length, area, volume).
An example is shown below.



9.1.4 Supports

Support is the intersection point between the body and the environment. There are may types of supports. Here are a few examples:

- Pivot/Pin: free rotation but constrained linear motions
- Roller: free linear motion in only one axis but constrained linear motion in other two axises.
- Fixed point: all motion constrained

9.1.5 Static equilibrium

An object is in static equilibrium state if net force and net torque on it are both zero.

$$\vec{F}_{net} = \sum F = 0$$
$$\vec{\tau}_{net} = \sum \tau = 0$$

9.2 Beam

A beam is a slender member that is capable of supporting load primarily by resisting against bending. The length of a beam is much greater than its width and height.

9.2.1 Axial deformation

When deforming along its neutral axis, the behaviours of beam can be characterized through Young's modulus.

Young's modulus E is defined as

$$E = \frac{Stress}{Strain} = \frac{\delta}{\epsilon}$$

, where strain is a measure of deformation and defined as

$$\epsilon = \frac{\delta L}{L}$$

and stress is a measure of force per unit area and defined as

$$\delta = \frac{F}{A}$$

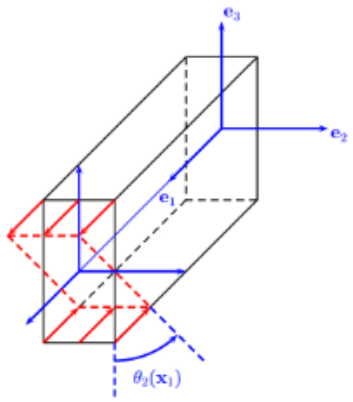
9.2.2 Non-axial deformation

When the beam is modeled as a 1D object and deformations occur non-axially, we can use **Euler–Bernoulli beam theory** to explain its behaviours.

Euler–Bernoulli beam theory

The theory has a few assumptions:

1. The deformations are small and therefore cross sections of the beam do not change in a significant manner
2. During deformation, the cross section of the beam remains planar and normal to the deformed axis of the beam, shown below.



With above assumptions, the relationship between the applied forces and the beam's deflection can then be captured by following formula:

$$\frac{d^2}{dx^2}(EI\frac{d^2w(x)}{dx^2}) = q(x)$$

, where w is the displacement between the beam axis's resting position and its new position after bending, q is the distributed load function, E is the Young's modulus of the beam and I is the area moment of inertia of the beam's cross section and is given by $\iint z^2\,dy\,dz$ assuming bending happens along z.

To solve this equation, one can use techniques such as "slope deflection method", "moment distribution method", and "moment area method."

Poison ratio, Shear modulus

When we model the beam as a 3D object and takes into account the effects of material deformations in directions other than the direction of the applied force, we may need Poison ratio and Shear modulus.

1. **Poison ratio** is defined as $\nu = -\frac{d\epsilon_{trans}}{d\epsilon_{axial}}$, where ϵ_{trans} is the non-axial strain and ϵ_{axial} is the axial strain. It describes the deformations in the directions perpendicular to the direction of force when a material is compressed in that direction, as shown below.

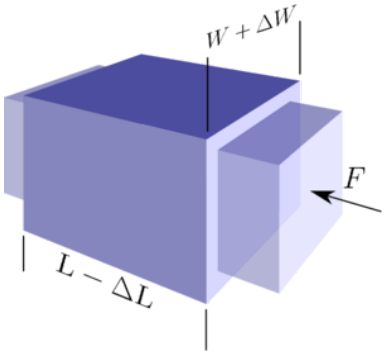


Figure 9.1: Poisson ratio. *From[5]*

2. **Shear modulus** is defined as $G = \frac{ShearStress}{ShearStrain} = \frac{F/A}{\Delta x/l}$, as shown in Fig 9.2.

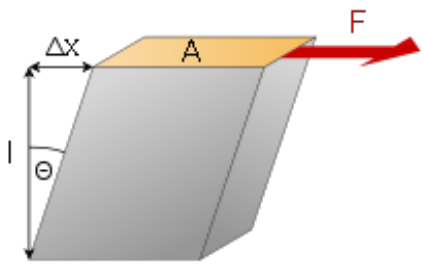


Figure 9.2: Shear modulus. *From[6]*

One may also need stress tensor and strain tensor to describe more complicated behaviors.

9.2.3 Mechanical failures

Yield

Yield happens when the material begins to deform plastically. After the yield point is passed, the deformation is non-reversible.

Creep

Creep happens when material deforms permanently even though the stress is still below the yield point. This may due to the result of long-term exposure to high levels of stress, and it may be affected by many other factors, such as material properties, applied distributed load and temperature.

Buckling

Buckling happens when a beam has a sudden sideways failure due to high compressions, which may still be less than the ultimate compressive stress that the beam can tolerate.

9.3 Reference

1. `https://en.wikipedia.org/wiki/Beam_(structure)`, 09/25/2016
2. `https://en.wikipedia.org/wiki/Euler%E2%80%93Bernoulli_beam_theory`, 10/02/2016
3. `http://web.mit.edu/16.20/homepage/7_SimpleBeamTheory/SimpleBeamTheory_files/module_7_no_solutions.pdf`, 10/02/2016
4. `http://subsurfwiki.org/wiki/Poisson's_ratio`, 10/02/2016
5. `https://en.wikipedia.org/wiki/Shear_modulus`, 10/02/2016

Lecture 10
Manipulation

Robotics: Design, Manufacturing, and Control **EE209AS – Fall 2016**
Prof. Ankur Mehta *Transcribed by: Tianrui Zhang*

10.1 Mobility/Manipulation Duality

In traditional **mobility** problems, the robot applies forces to earth, and the reaction forces move the robot.

On the contrary, in traditional **manipulation** problems, the robot is fixed to earth and applies forces to target object. The forces then move the object.

A harder type of problems is then to mingle the contradicting two to achieve **mobile manipulation**, which uses coordinated full body motions to effect target object and/or the environment. An examples is throwing a baseball. Here, we will only talk about manipulation problems.

10.2 Manipulation

In manipulations, **manipulator** is a type of device designed to manipulate target objects, which are often referred as **manipuland**. End effector is one example of manipulators.

There are many different ways of manipulations. One common way is through **pushing**. When pushing, the manipuland is not fully enclosed by the manipulator. The external force exerted by robot is only through non-slip contact between the robot and object.

Another common way is through **grasping**. In grasping, the manipuland is physically enclosed by robot manipulator, and then the robot applies forces to the manipuland.

10.2.1 Definition

In general, the manipulation problem can be defined as: given robot in configuration space C, set of obstacles in configuration space C_{obs}, the initial configuration q_i and the goal configuration q_f, find a path function x, which maps values from [0,1] to configuration space such that $x(0) = q_i, x(1) = q_f$, and x(s) $\notin C_{obs} \quad \forall s$ with additional two requirements:

1. The manipulator can't intersect with manipuland.
   The manipulator can't intersect with object.
   The manipuland can't intersect with object.

2. The manipuland must be stable and/or grasped in order to transfer

10.2.2 Steps of manipulation

The process of manipulation is usually involved with following steps:

1. **Perceive** the state of manipuland, such as its pose, type and properties

2. **Reach** the manipuland with the end effector
3. **Grasp/Push** the manipuland
4. **Move** the manipuland to its goal state

10.2.3 Grasping

In grasping problems, there are three main questions to be answered, namely:

1. Analysis: Given manipuland and grasp, is it closed?
2. Synthesis: Given manipuland, how find a closed graph?
3. Existance: Given manipuland, what is the minimum number of points for a closed grasp?

In general, there are two mian ways to attack these grasping problems: form closure and force closure.

Form closure

In form closure method, the idea is to contain the manipuland by geometry. Here, we present a 2D way to do it known as Reuleaux's method.

9.2.3.1.1 Reuleaux's method The method is used to determine if the manipuland is firmly grasped without tendency to rotate. It assumes that the manipulator can provide arbitary amount of force at contacting points.

The method is as follows:

1. Draw a perpendicular line at the contacting point
2. Put a $\oplus$ sign on the left plane to indicate tendency for counter-clockwise rotation, and put a $\ominus$ sign on the right plane to indicate tendency for clockwise rotation, as shown below.

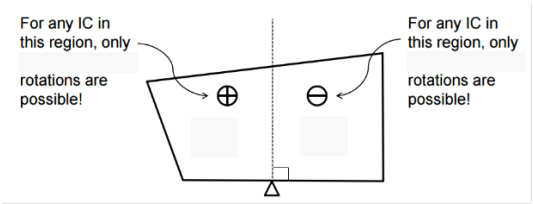


Figure 10.1: Reuleaux's method. *From[1]*

3. Repeat for each contacting point
4. If both signs exist in all regions created by those perpendicular lines, the grasp is firm and will not create rotations.

Example:

An example of grasping that will create rotation is shown below. Note that the central region has only $\ominus$ signs.

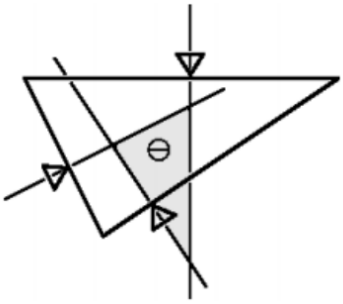


Figure 10.2: Reuleaux's method example. *From[1]*

Force closure

In force closure method, the idea is to contain the manipuland by force equilibrium. The consideration may or may not include friction forces.

To make the grasp firm, one should pick forces such that the net force and net torque are zeros.

$$\vec{F}_{net} = \sum F = 0$$
$$\vec{\tau}_{net} = \sum \tau = 0$$

10.3 Reference

1. `http://courses.csail.mit.edu/6.141/spring2015/pub/lectures/Lec14-ManipulationPartI.pdf`, 10/09/2016

Lecture 11
Navigation

Robotics: Design, Manufacturing, and Control **EE209AS – Fall 2016**
Prof. Ankur Mehta *Transcribed by: Tianrui Zhang*

11.1 Overview

Robot navigation usually involves several sub-parts: mapping, localization, and motion planning. We have already learned motion planning in previous lectures.
The navigation process often requires the use of many different types of sensors, especially ranging sensors, such as SONAR and LIDAR. Many times, cameras are also used when using computer vision techniques.

11.2 Mapping

The goal for robot mapping is to create the map or floor plan of the robot's environments. During mapping, the robot needs to estimate the state of obstacles given its own state.

In general, the process of mapping is the process of obtaining a collection of features of interest, and a representation of the geometric and topological relationships among them. Features can be multidimensional. For example, a 0D feature can be simply the existence of an obstacle at a location, and a 1D feature can be the outline of that obstacle.

11.2.1 Occupancy grids

A simple way to construct the map is through occupancy grids. The idea is similar to the discretization method in motion planning such that each grid accumulates evidence of presence of the obstacle. When robot is navigating in the environment, it updates the grids on-line with its recent measurements. The ranges returned from the obstacle implies grid intervals. An example is shown below:

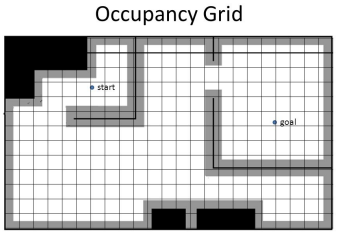


Figure 11.1: Occupancy grid example. *From[4]*

However, there are also problems created by this method. One common one is **quantization error**. The cell size chosen has a great influence on the final map. If the size is too big, the generated map may not be faithful to the robot's task. If the size is too small, the computation costs will be very expensive.

Another problem is **blurring**. It can be caused by pose estimation errors or sensor uncertainties.

11.3 Localization

The goal of robot localization is to locate the robot itself given the map or necessary information of its environments. During localization, the robot needs to estimate the state of itself with known obstacle state. Therefore, localization can be viewed as the reverse process of mapping.

11.3.1 Dead reckoning

If there is no **landmark** available in environment, **dead reckoning** method is usually the way to determine location of robot. The method uses proprioception to estimate robot's pose with respect to its initial coordinate frame, and this is usually done with robot's inertial measurement sensors, which typically includes accelerometer and gyroscope. From accelerometer, one can obtain distance that robot traveled. From gyroscope, one can obtain its orientation information. Therefore, two combined will give the robot trace.

11.3.2 Trilateration

If landmarks are available, **trilateration** method can be used to determine location of robot more accurately. The idea behind this method is that, given the distance between landmarks and the robot, the exact location can then be obtained by finding the intersection point of at least three spheres with landmarks at centers of these spheres. This is shown below.

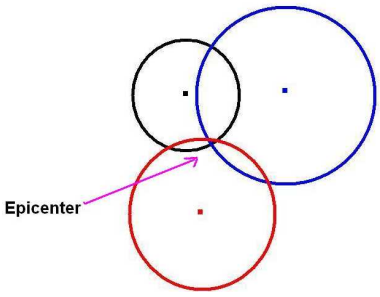


Figure 11.2: Trilateration method. *From[5]*

11.4 Simultaneous localization and mapping(SLAM)

If we combine mapping and localization together, we then have a **Simultaneous localization and mapping(SLAM)** problem. In SLAM, both robot state and obstacle state are unknown.
SLAM algorithm usually takes an incremental way to solve the problem. The robot first obtain its current state. Next, it move to a new robot state and acquire new observations. Lastly, it re-estimate its state, revise the map and use that as its new current state. This process is illustrated below.

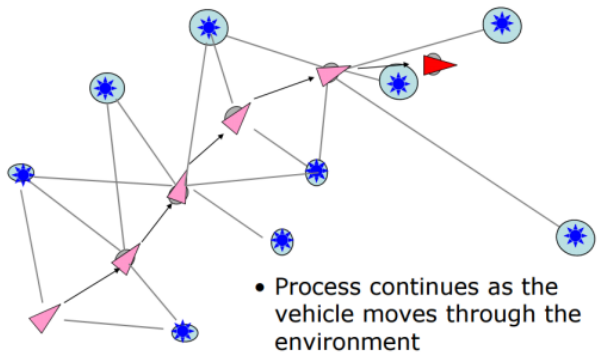


Figure 11.3: SLAM process. *From[3]*

SLAM is in general hard for many reasons. First because it is an interrelated problems between two seemingly contradicting problems, namely, localization and mapping. Next, it requires intelligence. Map making process is also hard for humans. Also, it may run into scaling issues. As the map grows, the memory space and disk space may have a combinatorial growth. Last but not the least, all uncertainties from sensors, actuators, or interpretation can contribute to the final errors.

11.5 Reference

1. `http://courses.csail.mit.edu/6.141/spring2015/pub/lectures/Lec13-Mapping.pdf`, 10/11/2016
2. `http://courses.csail.mit.edu/6.141/spring2015/pub/lectures/Lec12-Localization.pdf`, 10/11/2016
3. `http://courses.csail.mit.edu/6.141/spring2015/pub/lectures/Lec16-SLAM.pdf`, 10/11/2016
4. `http://images.slideplayer.com/23/6850840/slides/slide_3.jpg`, 10/11/2016
5. `http://www.coyleweb.net/triangulation.jpg`, 10/12/2016
6. `http://mathworld.wolfram.com/TriangulationPoint.html`, 11/2/2016

Lecture 12
System Engineering

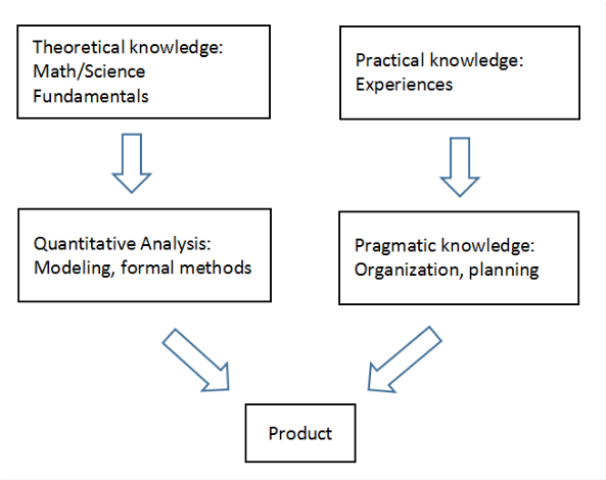
Robotics: Design, Manufacturing, and Control **EE209AS – Fall 2016**
Prof. Ankur Mehta *Transcribed by: Tianrui Zhang*

12.1 Overview

The purpose of this lecture is to provide you a holistic view of engineering from a practical standpoint and a toolkit of ideas to help you develop your own useful engineering practices.

Engineering is the process of specifying, designing, implementing, and validating physical artifacts with a desired set of properties, such as DOFs, appearance, and behaviors. Many times, this process involves decisions of **trade-offs**, such as weight vs. strength, power consumption vs. speed, and cost vs. lifetime.

Also, the engineering process often requires the use of theoretical and practical knowledge, and quantitative and pragmatic analysis. There relationships are shown below.



12.1.1 Elements to general engineering

- **Conception**:
  Develop a mental model of artifact, constraints, and assumptions about system environment
- **Execution**:
  Put mental model into practice, observe behavior, and compare to model-predicted behavior
- **Documentation**:
  Note down above two steps to match conception to execution as well as "Social contract", which

describes intent, trust that it happens.

12.2 Design

12.2.1 Hierarchical design

Hierarchical design is a design methodology that breaks large tasks into smaller subtasks and then identifies dependencies among theses subtasks. Below is an example.

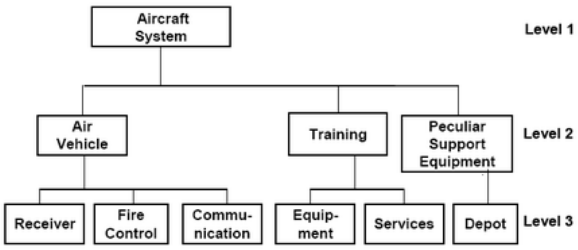


Figure 12.1: Hierarchical design example on aircraft system. *From[2]*

12.2.2 Iterative design

Another common design methodology is **iterative design**, which is a periodic process of designing, testing, analyzing, and redesigning the system based on the feedback until the design goals are achieved. It is not just simply try it and see the results.

This method often uses a technique called **Gantt chart**. Gantt charts indicate the begin and end dates of the subtasks of a project. Below is an example.

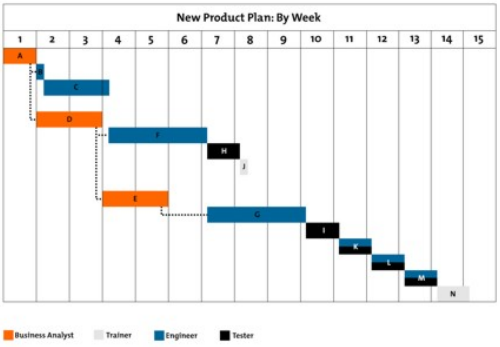


Figure 12.2: Gantt chart example. *From[3]*

12.3 Testing

12.3.1 Principles

Testing is a key component of engineering process. In principle, the testing process should be repeatable, systematic, and well documented.

Usually, testing phase adopts a **block box model**. The model says that testing engineer should only care about system outputs given system inputs without knowing how the testing block is implemented.

The combined test inputs and predicted outputs are usually called **test cases**. Also, these test cases should include boundary/edge cases as well as invalid inputs to make the testing more comprehensive. The testbench should also be written by different people, so use your teammates.

During testing, there are a few things to keep in mind.

1. Always know what is supposed to happen, write it down, and test for it.
2. Specify the preconditions, postconditions, and system invariants. This is often called "**Design by contract**."
3. Moreover, always remember to check for side effects, especially in debug codes.

12.3.2 Debugging

When debugging, visualization of system states is usually very helpful, because it can expose otherwise hidden system state and is very useful for communicating results.

12.4 Reference

1. `http://courses.csail.mit.edu/6.141/spring2015/pub/lectures/Lec07-SystemEngineeringAndTest.pdf`, 10/16/2016
2. `http://www.managementpedia.com/images/0/0f/WBS.png`, 10/16/2016
3. `https://www.mindtools.com/media/Diagrams/Gantt-Chart-Diagram-Figure-1small.jpg`, 10/16/2016

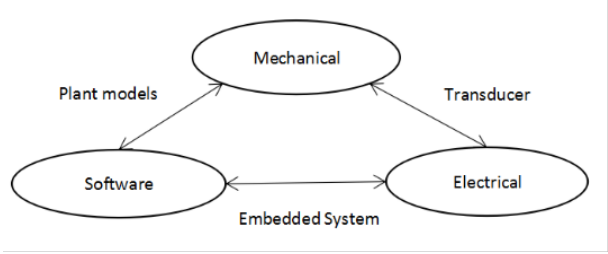
Lecture 13
Rapid design and prototyping

Robotics: Design, Manufacturing, and Control **EE209AS – Fall 2016**
Prof. Ankur Mehta *Transcribed by: Tianrui Zhang*

13.1 Design for manufacturing

Suppose that we have plans for our robotic system, now we need to physically realize a thing. Many times, we want to quickly validate our designs of robotic system before mass productions in the factory. **Rapid prototyping** provides exactly what we want here.

Typically, we do that by first separating electrical, manufacturing, software designs, and establishing constraints and dependencies among them. An example is shown below:



However, sometimes they can be deeply dependent, so not easy to separate. For example, the algorithms require controllers, which have physical size and power requirements and therefore imposes constraints on mechanical designs. In such cases, the goal is to abstract as much as possible to develop individual elements.

Then, we can begin to realize the system by rapidly prototyping each components.

13.2 Software prototyping

In software prototyping, we need to evaluate system models, algorithms, and interfacing first. Then, we can realize our software designs by using following commonly used prototyping languages and tools: Matlab, Python, Javascript, Octave, Blender, and Software Factory. These languages and tools are widely used because they are easier to use and usually have abundant library resources to support most computational needs.

In robotics field, a famous and widely used tool is **Robot Operating System (ROS)**[1].

One also needs to know the limitations of software prototyping, including memory models, data types, and computational limits.

13.3 Electronics prototyping

Electronics prototyping usually involves controller part and circuit part.

For controllers, commonly used embedded computers are Arduino, Raspberry Pi, BeagleBone Black and Intel Edison.

For circuits, people usually use breadboarding and PCB for prototyping.

13.4 Mechanical prototyping

The mechanical prototyping process should capture key mechanical attributes of your design, such as shape, appearance, and kinematic functions, and yet well balance the trade offs among time, cost, robustness and material properties.

13.4.1 3D printing

The most common mechanical prototyping method is 3d printing. One needs to design the mechanical structure model in CAD software first and let the 3D printer to create the structure. Common used CAD tools are SolidWorks, OpenSCAD and Blender.

There are many different types of 3D printing technologies.

Fused deposition modeling(FDM)

FDM works by laying down thermoplastic material in layers. The printer will melt the material first at nozzle and then squeeze a wire of such material out as the building block to form different shapes. The material then quickly cools down and solidified into objects, as shown below.

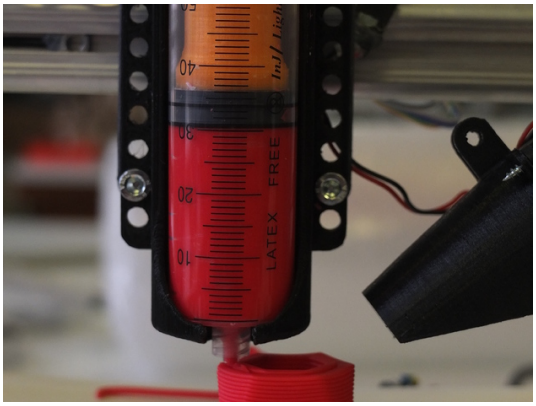


Figure 13.1: FDM printing. *From[2]*

The material used by FDM printingThe has a great influence on mechanical structure created, and therefore it may have problems, such as delamination and overhangs.

Stereolithography(SLA)

SLA works by shining an UV laser on to photopolymer materials. The material is then solidified and forms a single layer. Next, multiple layers will be formed based on a pre-programmed design from CAD tools, and the process is repeated until the 3D object is complete.

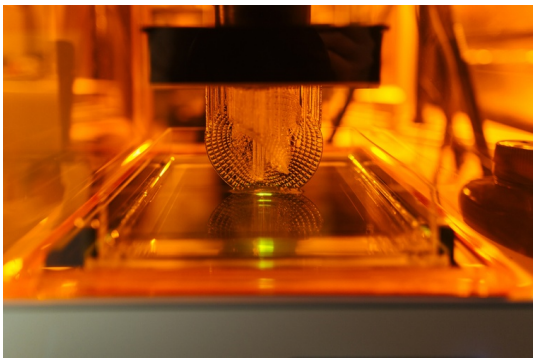


Figure 13.2: SLA printing. *From[3]*

Similar problems exist for SLA printing due to the material used, such as delamination and overhangs.

Selective laser sintering(SLS)

SLS works by focusing a high power laser to powdered material, typically metal, to sinter them together and form a single layer. Next, object is lowered and a new layer of powdered material is applied on top. The process then repeats to form more layers based on the CAD design.

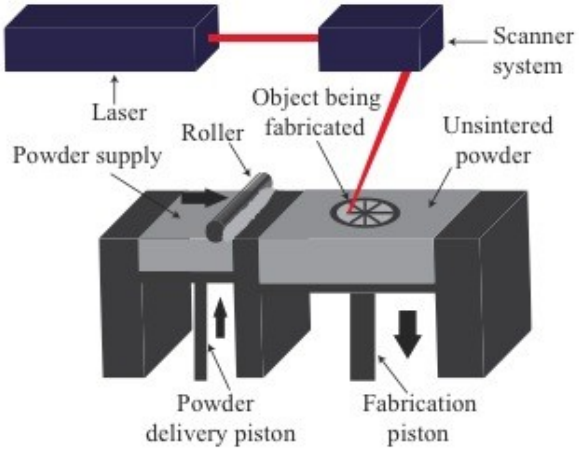


Figure 13.3: SLS printing. *From[4]*

SLS is usually very expensive and messy, and could be dangerous to use.

13.4.2 2D printing

Another common mechanical prototyping method is 2d printing. It works simply by cutting down 2D sheets with designed patterns, and them assemble the components into final shape, similar to origami.

Example
An example can be found at link 5.

13.5 Reference

1. `http://www.ros.org/`, 10/16/2016
2. `http://www.3ders.org/images2014/syringe-extruder-fdm-3d-printer-4.jpg`, 10/16/2016
3. `https://s3-eu-west-1.amazonaws.com/sdz-upload/prod/upload/3dprinting.jpg`, 10/16/2016
4. `http://cmrl.berkeley.edu/wp-content/uploads/RishiResearchAdditiveManufacturing-e1416951524639.jpg`, 10/16/2016
5. `https://www.youtube.com/watch?v=VFYJ74cUUIQ`, 10/16/2016

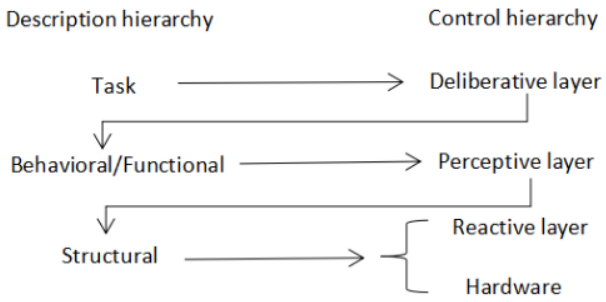
Lecture 14
Design Automation and modular robots

Robotics: Design, Manufacturing, and Control **EE209AS – Fall 2016**
Prof. Ankur Mehta *Transcribed by: Tianrui Zhang*

14.1 Design automation

The goal of design automation is to develop an automated process that takes high level task description of desired functionality and generates its physical realizations.

We can first break down task description into hierarchies and see how it is related to the control hierarchy we have learned.



From above, We can see that there is close relationship between description hierarchy and control hierarchy. Therefore, in principle, we can start with the high level task description and generate the deliberative layer of robotic system. Then, we can go one layer down to functional descriptions and pick the correspondent perceptive layer realizations. Lastly, we can choose reactive layer and hardware components from the structural description.

This process is similar to EDA(Electronic Design Automation) process in digital circuit design, where we have following layered automation process.

High level synthesis : C/C++ ->RTL language description (Verilog / VHDL)
Logic synthesis : RTL ->Netlist of logic gates
Layout : Netlist of logic gates ->Transistor-level layout

14.1.1 Implementations

To realize the first step that translates task description to deliberative layer, people have been creating high level **task description languages** similar to popular programming languages that have power to describe most of the robotic tasks. People are also trying to develop **domain specific languages(DSL)** that are good for a particular type of robotic tasks. This area is still under active researches.

```
align = composite_state:new {
 move_down = sista:new {
   entry = function () apply(move_down, {zt=0.01}) end
 },
 pre_align_z = simple_state:new {
   entry = function () apply(push_down, {zt=5}) end
 },
 align_z = simple_state:new {
   entry = function () apply(push_down, {zt=20}) end
 },
 transition:new { src='initial', tgt='move_down' },
 transition:new {
    src='move_down', tgt='pre_align_z',
    guard = function ()
              return get_force_TF().force.z > 4
            end },
 transition:new {
   src='pre_align_z', tgt='align_z',
   guard = function ()
             return get_move_duration() > 0.5
           end },
}
```

Figure 14.1: An example of Domain Specific Language for motion control. *From[1]*

To realize the following two steps, people have first created standard cells and design blocks at mechanical, electrical, and software level. People have even further combined them together to form integrated blocks. Then, the next step is to generate the final design from these standard cells. There are many ways to automate this combination process.

- **Human-in-the-loop**

  Human-in-the-loop design requires human interaction during the automation process. For example, we can ask people to help to tag each block, generate partial designs, or provide evaluations for the design, while still using other methods to automate the design process.

- **Brute force design**

  Brute force design simply does an exhaustive search on different combinations of the design blocks you have chosen. However, this method may not work well if the search space is large as the complexity will grow exponentially.

- **Data driven design**

  Data driven design uses the empirical data to speed up the design process. For example, one can pre-program the 2D unfolded version of a 3D model into the database, and then simply design the 3D model of the robot and let computer to finish rest of the work.[2]

- **Evolutionary design**

  Evolutionary design uses evolutionary algorithms to automate the design. The idea behind is based on Darwin's principle of natural selection. One can give the initial design of the robot and put it into a selective environment. By allowing the robot to "mutate" randomly in each iteration, the robot may eventually "evolve" into a complete design. An example can be found at link 3.

14.2 Modular robots

Modular robot is a type of robots that can be assembled by cell modules that have mechanical, electrical and/or software functionality. Therefore, they are able to change their shapes by rearranging each modules and easily adapt to new environment. This makes them very useful in many applications, such as space exploration and search and rescue missions.

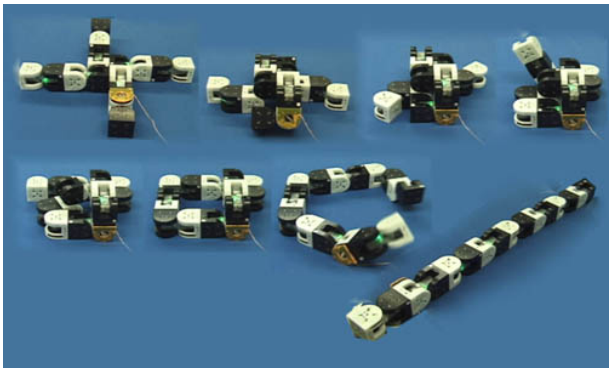


Figure 14.2: Modular robot. *From[4]*

Usually, these cell modules are designed in terms of standard cells, which typically has 1-DOF actuation ability. This modular characteristics has also made it especially suitable for design automation process.

Besides the designing algorithms mentioned above, one can also use tree-based search method for automated design of modular robots.

14.3 Reference

1. Markus Klotzbucher, Ruben Smits, Herman Bruyninckx, Joris De Schutter, "Reusable Hybrid Force-Velocity controlled Motion Specifications with executable Domain Specific Languages", 2011

2. Adriana Schulz,Cynthia Sung, Andrew Spielberg1, Wei Zhao1, Yu Cheng1, Ankur Mehta, Eitan Grinspun2, Daniela Rus1, Wojciech Matusik,"Interactive Robogami: Data-Driven Design for 3D Print and Fold Robots with Ground Locomotion", 2015

3. https://www.youtube.com/watch?v=z9ptOeByLA4, 10/18/2016

4. `https://www.smashingrobotics.com/wp-content/uploads/2012/07/wef34t-fergfsdt45tgdfg.jpg`,10/18/2016

Lecture 15
Multirobot systems

Robotics: Design, Manufacturing, and Control **EE209AS – Fall 2016**
Prof. Ankur Mehta *Transcribed by: Tianrui Zhang*

15.1 Overview

The goal of multirobot systems is to use multiple robots (nodes) that coordinate and cooperate to perform tasks that either one robot cannot, or multiple robots can perform more efficiently. For example, multirobot systems are often used in following fields: SLAM, navigation, manipulation, assembly, tracking, and area coverage in both gathering and deployment.

Multirobot systems contains several subparts. We will introduce each one below.

15.2 Cooperation and coordination

Cooperation in multirobot systems involves dividing up the highest level task into independent subtasks and then assigning them to different robots to perform each's tasks.

Coordination, on the contrary, involves combining forces from different robots in the system to accomplish a single task.

Cooperation and coordination of multiple robots can provide many more benefits than only using a single robot. For example, they can improve the overall system performance with expanded abilities, such as distributed sensing and actuation, or give higher fault tolerance to the system.

However, there will also be costs associated with using more than one robot. For example, the overal system costs may increase due to additional computational requirements, heavier communications burdens, and/or additional uncertainties and failures.

15.2.1 Metrics of importance

There are a few metrics to measure how well the multirobot system is performing the tasks.

- **Scalability**:

  Scalability measures the value (costs vs. benefits) in adding more robots into the system. For example, typical problems related to scalability issues are task distribution, computation, and communications.

- **Robustness**:

  Robustness measures how good the system can handle failures, such as loss of individual nodes or communication drop outs.

- **Modularity** :

  Modularity measures the degree to which the system can be separated and recombined, such as the replacability of individual nodes and the adaptability of nodes to changes in task environment.

15.2.2 Types of multirobot system

- **Type of robot**

  Based on type of robots used in the system, the multirobot system can be classified as homogeneous systems and heterogeneous systems.

  Homogeneous system involves robots that are alike in terms of many aspects such as their sizes and functionality, whereas heterogeneous system involves robots that are different in terms of those aspects.

  These characteristics make homogeneous system often very scalable but less optimal, and vice versa.

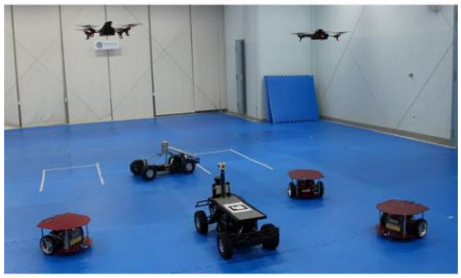


Figure 15.1: A heterogeneous multirobot system for SLAM. *From [1]*

- **Type of control**

  Based on type of controls used in the system, the multirobot system can be classified as decentralized/distributed systems and centralized systems.

  Distributed system is usually very robust, but centralized system is more provable and optimal.

  There are several centralized options. For instance, an central node can be responsible for any of the followings:

  1. **Sensing**: information fusion (similar) and integration (disparate)
  2. **Control**: coordinating actuation
  3. **Communication**: common channel

15.3 Communication

Communication is also a crucial part of multirobot system.

The objective of multirobot communication is to enable robots to exchange state information of robots themselves and their environment, or information between robots and users.

15.3.1 Taxonomy

There are many different aspects about multirobot communication. Below is a few of them.

- **Range**:infinite, nearby, or none
- **Bandwidth**: free vs. high , low vs. expensive, or zero vs. infinite
- **Medium**: optical, RF, acoustic, and/or chemical
- **Message distribution**: broadcast vs. point-to-point
- **Type**: explicit vs. implicit

15.3.2 Implicit and explicit communication

Implicit communication means communicating "through the world" such that each robot senses environmental state changes as a result of others' actions.

Explicit communication is to communicate through directed message passing.

The choice between implicit and explicit communication is usually a key aspect when designing the system. In general, the choice depends on the following four questions.

1. Is explicit communication needed at all?

   Explicit communication is usually not free. It not only requires power, hardware, design, processor cycles, and sometimes even licencing costs, but can cause additional point of failure as well. The information leakage from explicit communication is also another concern.

2. What should the information content be?

   In principal, one should use explicit communication for information that cannot be deduced from implicit ones.
   Depending on the task requirements, the information can be higher level goals or lower level state.
   For example, the content can be data exchange in exploration tasks, or negotiations for task assignments, and time synchronization information for sensing.

3. With whom should the information be shared?

   First off, disc model and hack model can be used to determine the candidates for information sharing. Next, the system may build different topologies for information sharing depending on the candidates, such as broadcast, addressed, tree, graph, DAG or mesh.

4. How to design such a system?

   When designing such a system, one should balance the trade offs among costs of communication, limited medium, and interference and noises from environment.

   The designer should also consider communication mechanism at different levels.

   - Physical layer: transmission medium, modulation, bitrate, signal coding, error correction, checksum
   - Link layer: time/frequency/code division, packet lost situations
   - Network layer: routing, dynamics
   - Higher layers: packetizing / framing, encryption, network coding

15.4 Flocking

The idea of flocking behavior in multirobot system is from the similar behaviour of birds in nature.

Basic models of flocking behavior are controlled by few simple rules:

- **Separation**: steer to avoid crowding neighbours
- **Alignment**: steer towards average heading
- **Cohesion**: steer towards average position
- **Go home**: steer towards "home"

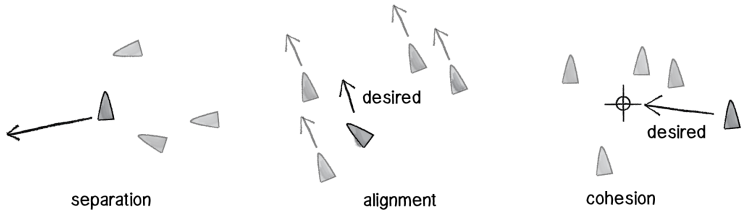


Figure 15.2: Boids flocking model. *From [2]*

With these few rules, the flock can form complicated emergent behaviours such as avoiding obstacles, safe wander, following the leader, and homing.

There are three ways to realize the flocking behaviour. The following table summarizes them and their costs.

Algorithms	Performance	Communication	Computation
Fully centralized control	high	high	high
Follow the leader	medium	medium	low
Follow neighbours	low	low	low

15.4.1 Task allocation

Task allocation is another important aspect of multirobot system. In general, task allocation is a problem that needs to map n atomic robots r_n to m different tasks t_m. The problem is NP-hard, and there are many challenges need to be taken into account, such as centralized vs distributed, online vs. offline, and instantaneous vs. time dependent.

15.5 Reference

1. `http://dronelife.com/2014/10/02/air-ground-based-autonomous-vehicles-working-together/`, 10/29/2016

2. `http://www.creativeapplications.net/processing/nature-of-code-by-daniel-shiffman-natural-systems-u`
`comment-page-1/`, 10/29/2016

Lecture 16
Underactuated and soft robotics

Robotics: Design, Manufacturing, and Control **EE209AS – Fall 2016**
Prof. Ankur Mehta *Transcribed by: Tianrui Zhang*

16.1 Underactuated system

The dynamics of mechanical systems are second order. Therefore, given configuration space vector q, its velocity vector $\dot{q}$, and system control input vector u, we then have the general form for a second-order controllable dynamical system:

$$\ddot{q} = f(q, \dot{q}, u, t)$$

Because the forward dynamics are mostly affine in commanded torque(From[1]), we can further decompose the above equation to

$$\ddot{q} = f_1(q, \dot{q}, t) + f_2(q, \dot{q}, t) * u$$

By using a technique called feedback linearization, we can set $u(q, \dot{q}, t) = f_2^{-1} * (r - f_1)$. Then, the above equation becomes

$$\ddot{q} = r$$

This means the control input can change the acceleration of robot in arbitrary directions, and therefore we can define any trajectory we want for the robot.

However, the above is only true when f_2 has full rank, that is, $rank(f_2) = dim(q)$. When $rank(f_2) < dim(q)$, we have an underactuated system.

As a result, we need to include system dynamics f_1 to get us to goal state. In fact, the control of underactuated systems still remains an open problem today and there are few general solutions for it.

16.1.1 Why underactuated system?

At first, underactuated system may sounds like a limitation, but it's actually highly liberating. The feedback linearization method is actually highly constraining.

In standard control theory, the controller needs to sense and estimate system state. It also need to make sure the system is operating within regions of high control authority. These all require energy and time.

By using an underactuated system, we may minimize energy usage and increase system speeds.

Some examples of common underactuated systems are acrobat, cart-pole, legged walking and aerial or underwater motions. For controls of these specific underactuated systems, please refer to link 2 below.

16.2 Soft robotics

Soft robot is a type of non rigid robots that have continuously deformable bodies. They are often built with soft and deformable materials like silicone, plastic, fabric, and rubber.

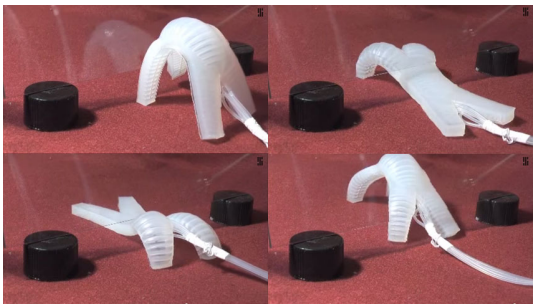


Figure 16.1: Soft robot example. *From[4]*

There are a number of advantages of using soft robots. Here, we listed a few.

- **Safety** is a huge advantage when using soft materials in the design of robots as softness allows safer physical contact with human bodies.
- **Maneuverability** is another advantage of soft robots over rigid ones, for the softness also allows robots to better handle uncertain and highly dynamic environments.
- **Robustness** is also an advantage because many key parts of rigid robots can be easily damaged, such as their joints.

However, soft robotics is also under active research these days, and their softness makes them difficult to model and hard to control.

To actuate a soft robot, one have many ways. Common ways include pneumatics, hydraulics, cables, and piezos.

To manufacture a soft robot, one can use casting or 3D printing.

16.3 Reference

1. `https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-832-underactuated-roboti readings/MIT6_832s09_read_ch01.pdf`, 10/29/2016
2. `https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-832-underactuated-roboti readings/`, 10/29/2016
3. `https://en.wikipedia.org/wiki/Soft_robotics`, 10/29/2016
4. `http://therobotblog.com/2014/09/10/four-legged-soft-robot-can-survive-being-crushed-by-car/`,10/29/2

Lecture 17
Human robot interaction

Robotics: Design, Manufacturing, and Control **EE209AS – Fall 2016**
Prof. Ankur Mehta *Transcribed by: Tianrui Zhang*

17.1 Overview

Human robot interaction is the study of interactions between humans and robots. It is similar to HCI(human computer interaction), but they are also quite different. HRI treats humans as an integral subsystem in the robotic system, while HCI is user centric.

Because of the uncertainty from human interactions, they both share many similar challenges:

1. Both are strongly heterogeneous system and need to handling disparate abilities
2. Both have noisy multi-modal communication
3. The end-users are difficult to model

17.2 HRI design tasks

In the designing phase of a HRI system, one should clearly understand the ultimate goals and costs of the system, each participants in the system, their capabilities and constraints, and what information is needed by whom.

For instance, the human may have requirements in strength, endurance, cognitive load, autonomy, and emotions, yet the robot may have its own requirements in strength, lifetime, communication bandwidth, and processing powers.

Next, in the implementation phase, the designer should partition the ultimate goal goal among the participants, and ensure that required information gets communicated between participants based on designs above. the designer should also identify and handle possible failure conditions for the systm.

Lastly, real testing is the key. Simulations can help, but humans are notoriously hard to simulate. There may be many unknown unknowns.

17.3 HRI subcategories

Depending on the different application scenarios, HRI can be further divided into subcategories.

17.3.1 Field and Service robots

Field and service robots field deals with doing parts of jobs that humans can't or don't want to do, and HRI for this field needs to consider how robot interacts with human in these unwanted scenarios.

Field robots need to be able to perform nontrivial tasks in a certain field environment. The field environment is an unconstrained environment, which can be dynamic and sometime even dangerous. Typical field robots include applications in agriculture, military, underwater, and self-driving cars.

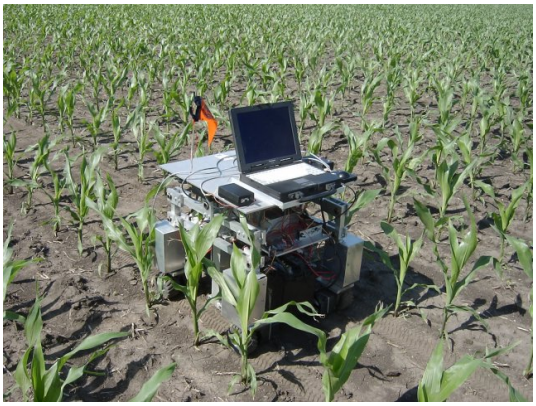


Figure 17.1: An agriculture field robot. *From[1]*

Service robots have a similar definition, but the service environment that they are working in is a relatively constrained environment. Examples of service robots are cleaning robots, such as iRobot cleaner.

17.3.2 Assistive robots

Assistive robot is designed to assist people to accomplish certain tasks, especially people with disabilities and seniors. Therefore, HRI in this field needs to incorporate people's needs for different tasks, such as senior people's physical needs.

Examples of assistive robot include humanoid robots for home healthcare and educations.

Figure 17.2: An assistive robot. *From[2]*

17.3.3 Social robots

Social robot needs to follow human social behaviors and rules. Therefore, HRI for social robots fields pays more attention to evaluating human emotions and social behaviors rather than robot dynamics or motions.

The ultimate goal is to integrate social robots into natural human interactions and let user forget that they are robots.

Examples of social robots include tour guide robots and robot pets.

Figure 17.3: Social robot Jibo. *From[3]*

17.4 Areas of research

The researches in HRI field are conducted in all three parts of sensing, planning, and acting.

17.4.1 Sensing and actuation

On sensing and actuation sides, the research is focused on the usage of different sensing techniques to interact with humans.

- **Touch: Haptics**

  The use of computer haptics is intuitive for motional tasks, such as remote surgery. The research is focused on force feedback and tactile sensing, such as distributed pressure and surface characteristics

- **Sight: Optics / Computer Vision**

  The use of computer vision techniques in this field can help robot recognize people's facial expressions and body languages.

- **Sound:**

  The use of sound signal in HRI is also a big field, as it is a natural way to interact with humans. Active researches in this field include voice recognition and generation and Natural Language Processing.

17.4.2 Control and planning

On control and planning sides, the research is focused on perceptions of humans behaviours.

- **Extrinsic communications**

  The research in extrinsic communications field pays attentions to perceive and signal human intentions, such as intentions of human drivers and pedestrians to self-driving cars.

17.5 Reference

1. `http://abe-research.illinois.edu/Faculty/grift/Research/BiosystemsAutomation/AgRobots/AgRobots.html`, 10/30/2016
2. `http://www.uta.edu/inquiry/winter14/highlights/robot-md.php`, 10/30/2016
3. `http://innoecho.com/personal-robotics-market-opportunities-and-business-models-33/`, 10/30/2016