Robotics Review

Saurabh Gupta
Robotic Tasks

Manipulation
Typical Robotics Pipeline

Observations ➞ State Estimation ➞ Planning ➞ Low-level Controller ➞ Control
Typical Robotics Pipeline

Observations → State Estimation → Planning → Low-level Controller → Control

Manipulation

Observed Images → 6DOF Pose → Grasp Motion Planning
Robot Navigation

Robot with a first person camera

Dropped into a novel environment

Navigate around

“Go 300 feet North, 400 feet East”

“Go Find a Chair”
Hartley and Zisserman. 2000. Multiple View Geometry in Computer Vision


Video Credits: Mur-Artal et al., Palmieri et al.
Typical Robotics Pipeline

1. Observations
2. State Estimation
3. Planning
4. Low-level Controller
5. Control

Observed Images

6DOF Pose

Geometric or Semantic Maps
Typical Robotics Pipeline

Observations → State Estimation → Planning → Low-level Controller → Control
Understand how to move a robot

Video from Deepak Pathak.
Terminology

- Link
- Joint
- End Effector
- Base
- Sensors

Slide from Dhiraj Gandhi.
Spaces

Work Space

Configuration Space

Task Space
Configuration Space

obstacles $\rightarrow$ configuration space obstacles

*Workspace*

(2 DOF: translation only, no rotation)

*Configuration Space*

Slide from Pieter Abbeel.
Configuration Space

Slide from Pieter Abbeel.
How to move your robot?

1. Task space to Configuration space

Initial configuration

Motion planning

Initial configuration

(x, y, θ)
Configuration Space to Task Space

Forward Kinematics

Slide from Dhiraj Gandhi.
Configuration Space to Task Space

Forward Kinematics

\[ P_x = l_1 \cos(\theta_1) + l_2 \cos(\theta_1 + \theta_2) \]
\[ P_y = l_2 \sin(\theta_1) + l_2 \sin(\theta_1 + \theta_2) \]
\[ P_\theta = \theta_1 + \theta_2 \]

Slide from Dhiraj Gandhi.
Configuration Space to Task Space

Forward Kinematics

Slide from Dhiraj Gandhi.
Configuration Space to Task Space

Forward Kinematics

\[ P_A = T_B^A P_B \]

\[
T_B^A = \begin{bmatrix}
    r_{11} & r_{21} & r_{31} & \Delta x \\
    r_{12} & r_{22} & r_{32} & \Delta y \\
    r_{13} & r_{23} & r_{33} & \Delta z \\
    0 & 0 & 0 & 1
\end{bmatrix}
\]
Configuration Space to Task Space

Forward Kinematics

\[ T_0^0(\theta_1) \ T_1^1(\theta_2) \ldots \ T_{n-1}^{n-2}(\theta_n) \ T_{n-1}^{n-1}(\theta_n) \]
Configuration Space to Task Space

Forward Kinematics

Maps configuration space to work space

\[
T = T_1^0(\theta_1) T_2^1(\theta_2) \ldots T_{n-1}^{n-2}(\theta_n) T_n^{n-1}(\theta_n)
\]

\[
x = f(\theta) = f(\theta_1, \theta_2, \ldots, \theta_{n-1}, \theta_n)
\]

Slide adapted from Dhiraj Gandhi.
Inverse Kinematics

Task Space to Configuration Space

Forward Kinematics

Inverse Kinematics

Numerical IK

Solve for $\theta_d$ in:

$$x_d - f(\theta_d) = 0$$

Analytical IK

- Robot Specific
- Fast
- Characterize the solution space

$x = f(\theta)$

Maps configuration space to work space

Find configuration(s) that map to a given work space point

Slide adapted from Dhiraj Gandhi, Modern Robotics
Task Space to Configuration Space

Analytical Inverse Kinematics

Slide from Dhiraj Gandhi.
How to move your robot?

1. Task space to Configuration space

Motion planning

Initial configuration

Desired configuration

$(x, y, \theta)$
How to move your robot?

1. Task space to Configuration space

2. Configuration space trajectory

Motion planning

Initial configuration

Desired configuration

\[(x, y, \theta)\]
Path Planning

Configuration Space With Obstacles + Initial Config. + Goal Config. → Feasible State Trajectory

Picture Credits: Palmieri et al.
Path Planning

1. Complete Methods
2. Grid Methods
3. Sampling Methods
4. Potential Fields
5. Trajectory Optimization
Probabilistic Roadmaps

Space $\mathbb{R}^n$  forbidden space  Free/feasible space
Probabilistic Roadmaps

Randomly Sample Configurations
Probabilistic Roadmaps

Randomly Sample Configurations

Slide from Pieter Abbeel.
Probabilistic Roadmaps

Test Sampled Configurations for Collisions
Probabilistic Roadmaps

The collision-free configurations are retained as milestones.
Probabilistic Roadmaps

Each milestone is linked by straight paths to its nearest neighbors
Probabilistic Roadmaps

Paths that undergo collisions are removed
Probabilistic Roadmaps

The collision-free links are retained as local paths to form the PRM
Probabilistic Roadmaps

The start and goal configurations are included as milestones.
Probabilistic Roadmaps

The PRM is searched for a path from $s$ to $g$
Probabilistic Roadmaps

Challenging to link milestones.
Probabilistic Roadmaps

Challenging to link milestones.

Collision checking can be slow.

All straight line paths may not be feasible, or a good measure of distance between states.
Rapidly Exploring Random Trees (RRTs)

**Kinodynamic planning**

Build up a tree through generating "next states" in the tree by executing random controls.
Rapidly Exploring Random Trees (RRTs)

Build up a tree through generating "next states" in the tree by executing random controls.

```plaintext
GENERATE_RRT(x_{init}, K, \Delta t)
1  \mathcal{T}.init(x_{init});
2  for k = 1 to K do
3    x_{rand} \leftarrow \text{RANDOM\_STATE}();
4  \quad x_{near} \leftarrow \text{NEAREST\_NEIGHBOR}(x_{rand}, \mathcal{T});
5  \quad u \leftarrow \text{SELECT\_INPUT}(x_{rand}, x_{near});
6  \quad x_{new} \leftarrow \text{NEW\_STATE}(x_{near}, u, \Delta t);
7  \quad \mathcal{T}.add\_vertex(x_{new});
8  \quad \mathcal{T}.add\_edge(x_{near}, x_{new}, u);
9  Return \mathcal{T}
```

**SELECT\_INPUT(x_{rand}, x_{near})**

- Two point boundary value problem
  - If too hard to solve, often just select best out of a set of control sequences.  
    This set could be random, or some well chosen set of primitives.

Slide from Pieter Abbeel.
Rapidly Exploring Random Trees (RRTs)

Build up a tree through generating "next states" in the tree by executing random controls.
Rapidly Exploring Random Trees (RRTs)

Build up a tree through generating "next states" in the tree by executing random controls.
How to move your robot?

1. Task space to Configuration space

2. Configuration space trajectory

Motion planning

Initial configuration

Desired configuration

\((x, y, \theta)\)
How to move your robot?

1. Task space to Configuration space

2. Configuration space trajectory

3. Trajectory execution

Motion planning

Initial configuration

Desired configuration

$(x, y, \theta)$
Trajectory Execution

Dynamically feasible trajectory $x_t^{ref}$ from planner

What control commands should I apply in order to get the robot to robustly track this trajectory?

Robot state $x_t$  Robot location, or joint angles.

Control sequence $u_t$  Velocities, torques.

Dynamics function $x_{t+1} = f(x_t, u_t)$  State evolution as we apply control.

Cost function $\sum_t \|x_t - x_t^{ref}\|$  

Low-level control can be formulated as an optimization problem.
**Trajectory Execution**

**Feedback Control**

Figure 11.1: (a) A typical robot control system. An inner control loop is used to help the amplifier and actuator to achieve the desired force or torque. For example, a DC motor amplifier in torque control mode may sense the current actually flowing through the motor and implement a local controller to better match the desired current, since the current is proportional to the torque produced by the motor. Alternatively the motor controller may directly sense the torque by using a strain gauge on the motor’s output gearing, and close a local torque-control loop using that feedback. (b) A simplified model with ideal sensors and a controller block that directly produces forces and torques. This assumes ideal behavior of the amplifier and actuator blocks in part (a). Not shown are the disturbance forces that can be injected before the dynamics block, or disturbance forces or motions injected after the dynamics block.

11.2 Error Dynamics

In this section we focus on the controlled dynamics of a single joint, as the concepts generalize easily to the case of a multi-joint robot.

If the desired joint position is $\mathbf{x}_d(t)$ and the actual joint position is $\mathbf{x}(t)$ then we define the joint error to be $\mathbf{x}_e(t) = \mathbf{x}_d(t) - \mathbf{x}(t)$.
Low-level Control

**Simplifying assumptions:**
Linear dynamics, quadratic cost.

⇒ **Linear Quadratic Regulator**
Exactly solved using dynamic programming.

The LQR setting assumes a linear dynamical system:

\[ x_{t+1} = Ax_t + Bu_t, \]

\(x_t\): state at time \(t\)
\(u_t\): input at time \(t\)

It assumes a quadratic cost function:

\[ g(x_t, u_t) = x_t^\top Q x_t + u_t^\top R u_t \]

with \(Q \succ 0, R \succ 0\).

For a square matrix \(X\) we have \(X \succ 0\) if and only if for all vectors \(z\) we have \(z^\top X z > 0\). Hence there is a non-zero cost for any state different from the all-zeros state, and any input different from the all-zeros input.
Linear Quadratic Regulator

Cost if the system is in state $x$, and we have $i$ steps to go.

$J_i(x)$

Cost if the system is in state $x$, and we have $i+1$ steps to go.

$J_{i+1}(x)$

$$= \min_u x^TQx + u^TRu + J_i(Ax + Bu)$$
Linear Quadratic Regulator

\[ J_{i+1}(x) \leftarrow \min_u \left[ x^\top Q x + u^\top R u + J_i(A x + B u) \right] \]

Initialize \( J_0(x) = x^\top P_0 x \).

\[
J_1(x) = \min_u \left[ x^\top Q x + u^\top R u + J_0(A x + B u) \right] \\
= \min_u \left[ x^\top Q x + u^\top R u + (A x + B u)^\top P_0(A x + B u) \right] \quad (1)
\]

To find the minimum over \( u \), we set the gradient w.r.t. \( u \) equal to zero:

\[
\nabla_u […] = 2Ru + 2B^\top P_0(A x + B u) = 0,
\]

hence:

\[
\begin{align*}
u &= -(R + B^\top P_0 B)^{-1}B^\top P_0 A x \\
&(2)
\end{align*}
\]

(2) into (1):

\[
J_1(x) = x^\top P_1 x
\]

for:

\[
\begin{align*}
P_1 &= Q + K_1^\top R K_1 + (A + BK_1)^\top P_0(A + BK_1) \\
K_1 &= -(R + B^\top P_0 B)^{-1}B^\top P_0 A.
\end{align*}
\]

Slide from Pieter Abbeel.
Linear Quadratic Regulator

In summary:

\[ J_0(x) = x^\top P_0 x \]
\[ x_{t+1} = Ax_t + Bu_t \]
\[ g(x, u) = u^\top Ru + x^\top Qx \]

\[ J_1(x) = x^\top P_1 x \]
for: \[ P_1 = Q + K_1^\top RK_1 + (A + BK_1)^\top P_0 (A + BK_1) \]
\[ K_1 = -(R + B^\top P_0 B)^{-1} B^\top P_0 A. \]

\( J_1(x) \) is quadratic, just like \( J_0(x) \).

Update is the same for all times and can be done in closed form for this particular continuous state-space system and cost!

\[ J_2(x) = x^\top P_2 x \]
for: \[ P_2 = Q + K_2^\top RK_2 + (A + BK_2)^\top P_1 (A + BK_2) \]
\[ K_2 = -(R + B^\top P_1 B)^{-1} B^\top P_1 A. \]
Linear Quadratic Regulator

Set $P_0 = 0$.
for $i = 1, 2, 3, \ldots$

$$K_i = -(R + B^T P_{i-1} B)^{-1} B^T P_{i-1} A$$

$$P_i = Q + K_i^T R K_i + (A + BK_i)^T P_{i-1} (A + BK_i)$$

The optimal policy for a $i$-step horizon is given by:

$$\pi(x) = K_i x$$

The cost-to-go function for a $i$-step horizon is given by:

$$J_i(x) = x^T P_i x.$$
Linear Quadratic Regulator

Extensions which make it more generally applicable:

- Affine systems System with stochasticity
- Regulation around non-zero fixed point for non-linear systems
- Penalization for change in control inputs
- Linear time varying (LTV) systems
- Trajectory following for non-linear systems
Linear Quadratic Regulator
How to move your robot?

1. Task space to Configuration space

2. Configuration space trajectory

3. Trajectory tracking

Motion planning

Initial configuration

Desired configuration

Initial configuration

Desired configuration

$(x, y, \theta)$

Robot setup with objects
Minor Detail

Camera Calibration

Images from Lerrel Pinto.
Camera Calibration

Slide from Dhiraj Gandhi.
Camera Calibration

$T^B_C$?
Camera Calibration

\[
\min_{T_C^B} (\| X_B - T_C^B X_C \|)
\]
Camera Calibration

\[ \min_{T_C^B} \sum_{i=1}^{n} \left\| X_B^i - T_C^B X_C^i \right\| \]

Slide from Dhiraj Gandhi.
Good Softwares

Movelt!

```
<robot name="baxter">
  <link name="base">
  </link>
  <link name="torso">
    <visual>
      <origin rpy="0 0 0" xyz="0 0 0"/>
      <geometry>
        <mesh filename="package://baxter_description/meshes/torso/base_link.DAE"/>
      </geometry>
    </visual>
    <collision>
      <origin rpy="0 0 0" xyz="0 0 0"/>
      <geometry>
        <mesh filename="package://baxter_description/meshes/torso/base_link_col"/>
      </geometry>
    </collision>
  </link>
</robot>
```

Slide from Dhiraj Gandhi.
MoveIt! Example

target_poses = [
    {'position': np.array([0.28, 0.17, 0.22]),
     'pitch': 0.5,
     'numerical': False},
    {'position': np.array([0.28, -0.17, 0.22]),
     'pitch': 0.5,
     'roll': 0.5,
     'numerical': False}
]

robot.arm.go_home()

for pose in target_poses:
    robot.arm.set_ee_pose_pitch_roll(**pose)
    time.sleep(1)
robot.arm.go_home()
Robotics Review: How to move your robot?

1. Task space to Configuration space
2. Configuration space trajectory
3. Trajectory execution

Configuration Space
Forward / Inverse Kinematics
Motion Planning
Optimal Control

Motion planning

Initial configuration
Desired configuration

(x, y, θ)
Resources

Kris Hauser's Robotic Systems Book
http://motion.cs.illinois.edu/RoboticSystems/

Pieter Abbeel's Advanced Robotics Course at Berkeley
https://people.eecs.berkeley.edu/~pabbeel/cs287-fa19/

Howie Choset's Robotic Motion Planning Course at CMU
https://www.cs.cmu.edu/~motionplanning/