Rigid Body Transformations

1. Preserves Length ($\forall p, q \in \mathbb{R}^3$), $||g(p) - g(q)|| = ||p - q||$
   - $\|R_{ab}(p_b - p_a)\|^2 = (\cdot)^\top(\cdot) = \|p_b - p_a\|^2$
2. Preserves relative Orientations between vectors:
   - $(\forall p, q \in \mathbb{R}^3)$, $g(p) \times g(q) = g(p \times q)$
3. Preserves coordinate frames

- Rigid body transformations preserve inner products
  - $x \cdot y = g(x) \cdot g(y)$
- Rotations are rigid body transformations
- $q_a = R_{ab} q_b$ is transferring from B to A
- g_{ab} means present frame B in frame A (using A's basis to describe B)

Law of Cosines: $c^2 = a^2 + b^2 - 2ab\cos(\theta_C)$
where a, b, c are side lengths of the triangle.

$$e^{\mathbf{A}} = \sum_{n=0}^{\infty} \frac{\mathbf{A}^n}{n!} = \mathbf{I} + \mathbf{A} + \frac{1}{2!}\mathbf{A}^2 + \cdots$$

$(e^{\mathbf{A}})^\top = e^{(\mathbf{A}^\top)} = e^{-\mathbf{A}}$ (if A is skew symmetric)
$e^{A+B} = e^A e^B$ for any square matrices $\mathbf{A}$, $\mathbf{B}$ such that $\mathbf{AB} = \mathbf{BA}$
g is any invertible square matrix of the same size as $\mathbf{A}$. Then $e^{g\mathbf{A}g^{-1}} = ge^{\mathbf{A}}g^{-1}$. $\mathbf{A}\vec{v} = \lambda\vec{v} \implies e^{\mathbf{A}}\vec{v} = e^{\lambda}\vec{v}$. $\det(A) = \det(A^T)$
$\det(e^{\mathbf{A}}) = e^{\lambda_1}e^{\lambda_2}\cdots e^{\lambda_n} = e^{\lambda_1+\lambda_2+\lambda_n} = e^{\mathrm{tr}\,\mathbf{A}} \implies \exists \exp(\cdot)^{-1}$.
$(AB)^T = B^T A^T, (AB)^{-1} = B^{-1}A^{-1}, (A+B)^T = (A^T + B^T)$
All matrices are associative: A(BC) = (AB)C but not necessarily commutative: $AB \neq BA$. $\det(R) = \pm 1, \|Ru\| = \|u\|$, for orthog matrix R, $\vec{u}$. $(e^{\mathbf{A}})^{-1} = e^{-\mathbf{A}}$
$\frac{\mathrm{d}x}{\mathrm{d}t} = ax(t)$, for $t \geq 0, a \in \mathbb{R}$, assuming the initial condition $x(0) = x_0$. And $x(t) > 0 \forall t \in \mathbb{R}$. $\therefore x(t) = x_0 e^{at}$
For any linearly independent set of vectors, we can pick a basis for those vectors which makes the set orthonormal.
Determinants are continuous. Plug in an easy value like 0 to compute at 1 point, then can conclude about other points.

Solving Matrix Diff eq's:

$\frac{\mathrm{d}x}{\mathrm{d}t} = \mathbf{A}x(t)$

1. Find eigenvalues via sols to $\det(\mathbf{A} - \lambda\mathbf{I}) = 0$.
2. Find eigenvectors by solving $\mathbf{A}\vec{v} = \lambda\vec{v}$ for eigvec $\vec{v}$.
3. Diagonalize $\mathbf{A} = \mathbf{V}\Lambda\mathbf{V}^{-1}$, where $V = \begin{bmatrix}\vec{v}_1 & \vec{v}_1\end{bmatrix}$
4. Let $\tilde{x} \triangleq V^{-1}x \implies \frac{\mathrm{d}\tilde{x}}{\mathrm{d}t} = \mathbf{\Lambda}\tilde{x}(t)$
5. Each row reduces to a scalar diffeq with known solution!

Submultiplicative if for all $A, B \in \mathbb{R}^{n\times n}$, $\|AB\| \leq \|A\| \cdot \|B\|$.
Frobenius Norm: $\|A\|_F = \sqrt{\mathrm{Trace}\,(AA^T)}$ is submultiplicative. The trace of any square matrix = sum of its eigenvalues.
Cross product formula:

$$\vec{a} \times \vec{b} = \begin{vmatrix} i & j & k \\ a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{vmatrix} = \begin{bmatrix} a_2b_3 - a_3b_2 \\ a_3b_1 - a_1b_3 \\ a_1b_2 - a_2b_1 \end{bmatrix}$$

Symmetric Matrix: $A = A^T$; Skew-Symmetric Matrix: $A = -A^T$
Diagonal elements must be 0 (otherwise like 5 cannot equal -5)

Rotation Matrices:

Transformation Matrices are **valid Rotation Matrices** if:

1. $\mathbf{R}^\top\mathbf{R} = \mathbf{R}\mathbf{R}^\top = \mathbf{I}$
2. $\det(R) = +1$, $\det(R^TR) = \det(R^T)\det(R) = \det(R)^2 = 1$

Examples: $R_X(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{bmatrix} = e^{\hat{x}\theta}, R_Y(\theta) = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix}, R_Z(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$

Writing vectors in frame B and frame A:
$v_b = v_{bx}x_b + v_{by}y_b + v_{bz}z_b$
$v_a = v_{bx}x_{ab} + v_{by}y_{ab} + v_{bz}z_{ab}$
Thus $v_a = R_{ab}v_b$
RPY (right to left, fixed frame) vs Euler (left to right, uses new axes)

Definition of a Group:

1. Closure:$g_1, g_2 \in G, g_1 \cdot g_2 \in G$
2. Identity: $\exists I \in SO(3) : R \cdot I \in SO(3)$, $R \cdot I = I \cdot R = R$
3. Inverse element: $\forall R \in SO(3), R^{-1} = R^T$, $R^{-1} \cdot R = I = R \cdot R^{-1}$
4. Associativity: $R_1(R_2R_3) = (R_1R_2)R_3$

Common Groups:

$SO(3) := \{R \in \mathbb{R}^{3x3} \mid R^TR = I,\ \det(R) = 1\} \subseteq \mathbb{R}^{3\times 3}$
$so(3) := \{A \in \mathbb{R}^{3x3} \mid A = -A^T\} \subset \mathbb{R}^{3\times 3}$ = skew-symmetric mat.'s
$SE(3) := \{(R, p) | R \in SO(3), p \in \mathbb{R}^3\}$
= Set of all pairs of rotation matrices and translations

$se(3) := \left\{\hat{\xi} = \begin{bmatrix} \hat{\omega} & v \\ 0 & 0 \end{bmatrix} | \hat{\omega} \in so(3), v \in \mathbb{R}^3\right\} \subset \mathbb{R}^{4\times 4}$
= set of all twist matrices $\hat{\xi}$

$g \in SE(3) = \begin{bmatrix} R & p \\ \vec{0}^\top & 1 \end{bmatrix} \in \mathbb{R}^{4\times 4}$ = set of all rigid body transformations where $R \in SO(3)$ and $p \in \mathbb{R}^3$.

$g^{-1} = \begin{bmatrix} R^T & -R^Tp \\ \vec{0}^\top & 1 \end{bmatrix} \in SE(3)$, $\hat{\omega} = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix}$

Hat map: $\wedge : \mathbb{R}^3 \to so(3)$.
Alternatively $a \times b = a^\wedge b$
Hat map properties: For all $R \in SO(3)$ and all $v, w \in \mathbb{R}^3$ the following hold:

$$R\hat{w}R^T = (Rw)^\wedge$$
$$R(v \times w) = (Rv) \times (Rw)$$

$\omega_1^2 + \omega_2^2 + \omega_3^2 = 1 \iff \|\omega\| = 1, \hat{\omega}^T = -\hat{\omega}$
$\det(A) = (-1)^n \det(A)$, odd-dimensioned skew-sym have det = 0

Rodrigues' formula ($\|\omega\| = 1$):

- $e^{\hat{\omega}\theta} = I + \hat{\omega}\sin\theta + \hat{\omega}^2(1 - \cos\theta)$
- $\hat{\omega}^2 = \omega\omega^T - I$, $\hat{\omega}^3 = -\hat{\omega}, \hat{\omega}^4 = -\hat{\omega}^2, \hat{\omega}^5 = -\hat{\omega}$

Homogeneous Representation: $g \in SE(3)$

- Points: $q = \begin{bmatrix} q_1 & q_2 & q_3 & 1 \end{bmatrix}^\top$
- Vectors: $\vec{v} = p - q = \begin{bmatrix} v_1 & v_2 & v_3 & 0 \end{bmatrix}^\top$

Exponentials of skew symmetric matrices produce elements of $SO(3)$: Given any unit vector $\omega \in \mathbb{R}^3$ and any scalar $\theta \in \mathbb{R}$: $e^{\hat{\omega}\theta} \in SO(3)$. XYZ $\mapsto$ Roll Pitch Yaw.
Euler's Theorem: Any rotation or orientation $R \in SO(3)$ is equivalent to a rotation about an axis ω through an angle θ.
Twist Coord: $\xi \in \mathbb{R}^6$ where $\xi = \begin{bmatrix} v \\ \omega \end{bmatrix}$ where $\omega \in \mathbb{R}^3$ is the axis of rot.
Note that the constant vector $v \neq$ linear velocity (as linear velocity is always changing) although it includes info about it
Cases:

1. General Screw Motion == Screw Joint
   - $v = -\omega \times q + h\omega$, where h is pitch $= \frac{\text{trans}}{\text{rot}} = \frac{d}{\theta}$ and q is some point from some other reference frame
   - $\xi = \begin{bmatrix} -\omega \times q + h\omega \\ \omega \end{bmatrix}$
2. Pure Rotational Motion == Revolute Joint
   - $\xi = \begin{bmatrix} -\omega \times q \\ \omega \end{bmatrix}, v = -\omega \times q$ as $h = 0 \implies h\omega = 0$
3. Pure Translational Motion == Prismatic Joint
   - $\xi = \begin{bmatrix} v \\ 0 \end{bmatrix}, v =$ direction of screw motion, $\omega = 0$

For 1 and 2, ω is a unit vector For 3, v is a unit vector
Chasles' Theorem: Every rigid body motion may be represented by a screw motion – a rotation about an axis followed by a translation parallel to that axis.
The transformation g corresponding to S has the effect on point p:

- $gp = q + e^{\hat{\omega}\theta}(p - q) + h\theta\omega$

Joint Space:

- Set of all possible joint positions for our basic joints
- $Q = S^1 \times S^1 \times R$ where "x" is the cartesian set product
- $Q = \theta_1 x \theta_2 x \theta_3$

Fwd kinematics mapping: $g_{st}(\theta) : Q \to SE(3)$. Expo Coords: (ξ, θ)
Wedge: $\hat{\xi} = \begin{bmatrix} v \\ w \end{bmatrix}^\wedge = \begin{bmatrix} \hat{\omega} & v \\ 0 & 0 \end{bmatrix}$

1. $e^{\hat{\xi}\theta} = \begin{bmatrix} I & v\theta \\ 0 & 1 \end{bmatrix}$, if ω=0
2. $e^{\hat{\xi}\theta} = \begin{bmatrix} e^{\hat{\omega}\theta} & (I - e^{\hat{\omega}\theta})(\omega\times v) + \omega\omega^T v\theta \\ 0 & 1 \end{bmatrix} (Twist) = \begin{bmatrix} e^{\hat{\omega}\theta} & (I - e^{\hat{\omega}\theta})q + h\theta\omega \\ 0 & 1 \end{bmatrix} (Screw)$, if $\omega \neq 0$, $||\omega|| = 1$

The degrees of freedom (DOF) of a robot arm are the number of joints in the robot which we may independently move.
Files Breakdown:

- ~/.bashrc: sets environment variables when running script to find ROS-specific things
- Build: stores information for building packages
- Devel: automatically generated files (e.g. header files)
- Src: Source code for packages (includes CMakeList.txt, Package.xml, include files)
- CMakeList.txt: input to cmake build system for building software packages
- Package.xml: metadata about the package contents/configuration and dependencies

- Include: C++ include headers

Essential Commands:

- **catkin_make**, then **source devel/setup.bash**
- MUST RUN AFTER DONE WITH LAB: ctrl+C out of all terminals, then pkill -u [username]
- roscd, rosls, **roscore** starts a server that all ROS nodes use to communicate

Nodes: Processes that compute

- an executable that uses ROS to communicate w/ other nodes
- **rosnode info /node_name**; No **cd**: **rosrun pkg script.py args**; **rospy.init_node(node_name, anon: unique name)**
- package.xml: build_depend, run_depend metadata
- rospack find package_name
- sourcing adds path of package to ROS_PACKAGE_PATH
- in **/src**, **catkin_create_pkg <package_name> <list_of_dependencies>**
- lauch file specifies several nodes to launch: roslaunch package_name launch_file.launch

Topics: Queues over which nodes exchange messages

- Nodes can publish messages to a topic as well as subscribe to a topic to receive messages
- **rostopic echo /turtle1/cmd_vel**, echo the message that a node is publishing to the topic **/turtle1/cmd_vel**; this creates a new node in cqt_graph
- **/teleop_turtle** publishes a message on topic **/turtle1/cmd_vel**, and the node **/turtlesim** subscribes to the topic to receive the message.

Services:

- service, request, and response types
- **rosservice type /clear** check the type of **/clear** service
- specific datatypes and args of requests and responses can be found in /srv/service.srv file
- **rosservice call [service] [arguments]** use **rosservice call** command to run

Message Types (ex. **std_msgs/String**):

- a ROS datatype used to exchange data between nodes
- variables and types in /msg/Message.msg of package; need to update package.xml and CMakeLists.txt after creation
- usage: from package_name.msg import message_name. Try to make message name different from package name

Publisher: Node that sends message to a topic

- **define talker()**: method which contains the node's main functionality
- **pub = rospy.Publisher('[topicName]', [msgType], queueSize = 10)**
- **r = rospy.Rate(10)**: publish at 10Hz publishing rate
- **while not rospy.isShutdown():**
```
    pub.publish(pubString)
    r.sleep()
```

Subscriber: Node that receives message from a topic

- def callback(message): called whenever this node receives a message on the subscribed topic, received message is 1st argument
- listener(): contains node's main functionality
- rospy.Subscriber([topicName], [MsgType], callback)
- rospy.spin()

Server:

- **rospy.Service(**
  '/{}/patrol'.format(sth = sys.argv[1]), # Service name
  Patrol, # Service type
  patrol_callback) # Service callback
- **rospy.wait_for_service(<service name>)**
- **serv = rospy.ServiceProxy('service name','service type') serv(<args>)** creates a proxy serv that we can send requests to

Client:

- **rospy.wait_for_service('[service name]')**
- **patrol_proxy = rospy.ServiceProxy(**
  [service name]('/xx/xx'), [service type](Patrol))

Mobile Robots TF:

```
rosrun tf tf_echo <source frame> <target frame> prints
info about a transformation between the two frames
tfBuffer = tf2_ros.Buffer()
tfListener = tf2_ros.TransformListener(tfBuffer)
trans = tfBuffer.lookup_transform(turtlebot_frame,
goal_frame, rospy.Time())
control_command.linear.x = K1*trans.transform.translation.x
control_command.angular.z= K2*trans.transform.translation.y
```

Computer Vision

Example:

$$K = I,\ \lambda_1 x_1 = KX_1,\ \lambda_2 x_2 = KX_2 \implies \boxed{\lambda_1 x_1 = X_1,\quad \lambda_2 x_2 = X_2}$$

$$g_{21} = (R, T), X_2 = RX_1 + T$$

normalized: means the focal length is 1 **Grayscale:** 0 (pure black) and 255 (pure white) **Thresholding:** binary image (not grayscale) with 1's (white) representing our foreground or object of interest, and 0's (black) **Two-View Geometry:** there's the **epipolar constraint** which has $(x')^T Ex = 0, E = \hat{T}R$, E is the essential matrix, T and R are applied on x to get x', this allows us to get depth
homography: apply affine transformation to an image t ochange perspectives (can straighten a pic), needs at least 4 pairs of points to do this
Launchfile format:

```
<launch>
<param name="marker_size" default="16.5">
<node>
<node name="" pkg="" type="" output="">
    <param name="" type="" value="$(arg marker_size)">
</node>
</launch>
```

Parameter server: ROS parameters are key-value pairs that ROS allows you to specify when launching a nodes that may be queried by those nodes at run-time.

Adjoints

- $(Ad_g)^{-1} = Ad_{g^{-1}}$ for all $g \in SE(3)$
- $Ad_{g_1 g_2} = Ad_{g_1} Ad_{g_2}$ for all $g_1, g_2 \in SE(3)$

Action File .action file that specifics the data of all the action servers $move_group$ action server sends out feedback and result msgs while the action client recives these msgs and sends out a goal msg
Velocities

- $q_a(t) = g_{ab} q_b \implies \dot{q}_a(t) = \dot{g}_{ab} q_b \implies v_{q_a}(t) = \dot{g}_{ab} g_{ab}^{-1} q_a$
- **Circle Method:** for v^s measure the length from joint to A-frame,this is your radius, radius * theta dot = velocity (make a circle by rotating the line), for v^b measure length from joint to body frame instead make a circle by rotating this radius

Dynamics

$$T(\theta, \dot{\theta}) = \sum_{i=1}^{n} T_i(\theta, \dot{\theta}) = \frac{1}{2}\dot{\theta}^T \left(\sum_{i=1}^{n} J_i^{b^T} M_i^b J_i^b\right)\dot{\theta}$$

$$M(\theta) = M^T(\theta), \dot{\theta}^T M(\theta)\dot{\theta} \geq 0, \dot{\theta}^T M(\theta)\dot{\theta} \iff \dot{\theta} = 0$$

$$V(\theta) = \sum_{i=1}^{n} m_i g h_i(\theta)$$

- $\dot{M} - 2C$ is skew-symmetric
- when computing J_i remember the correct adjoints and ξ_i for each jacobian, some jacobians might have cols with 0s cause not enough joints
- Coriolis matrix represents fictional forces caused by rotating basis vectors, (i.e. coriolis and centrifugal force)
- **when picking generalized state pick something and freeze it, if anything else in the system is still able to move then pick a variable that expresses that and freeze it as well, if everything is frozen then you're good to go!**, x found with springs a lot, θ most definitely for rotations
- **Task Space Dynamics:** if $x = f(\theta) \implies \dot{x} = \frac{\delta f}{\delta \theta}\dot{\theta} = J\dot{\theta} \implies \ddot{x} = J\ddot{\theta} + \dot{J}\dot{\theta}$ and $\Gamma = J^T F$, F is the force we can control to move the tool $F = \tilde{M}\ddot{x} + \tilde{C}\dot{x} + \tilde{N}$ (you then solve this the same way as computed torque but using x instead of θ) for ex: $F = \tilde{M}(\ddot{x}^d - k_p e_x - k_d \dot{e}_x)$**TO GET FULL EXPRESSION ISOLATE $\ddot{\theta}$ IN JOINT SPACE DYNAMICS AND PLUG IT INTO the equation for** $\ddot{x}$

Control

- **Joint-Space Control:** *given:* $\theta^d(t)$ desired joint trajectory
  goal: $\theta(t) \to \theta^d(t)$ s.t. $\lim_{t\to\infty} \|\theta^d(t) - \theta(t)\| = 0$
- **Task-Space Control:** *given:* $g^d_{ST}(t)$ desired tool trajectory,
  goal: design joint torque τ s.t. $lim_{t\to\infty} g_{ST}(t) \to g^d_{ST}(t)$
  USE computed-torque control!
- **Computed-Torque Control:** Choose Γ s.t. $\theta(t) \to \theta^d(t)$ as $t \to \infty$ can be done by having the scenario
- We want $\ddot{e} + K_d\dot{e} + K_p e = 0$, $e = (\theta^d(t) - \theta(t)$ this can be done by picking a Γ s.t.
  $M(\theta)\ddot{\theta} + C(\theta,\dot{\theta})\dot{\theta} + N(\theta) - \Gamma = \ddot{e} + K_d\dot{e} + K_p e = 0$
- Commonly found that
  $\Gamma = M(\theta)\ddot{\theta}^d + C(\theta,\dot{\theta})\dot{\theta} + N(\theta) + M(\theta)(-K_p e - K_d\dot{e})$
  $$\begin{bmatrix}\dot{e}\\ \ddot{e}\end{bmatrix} = \begin{bmatrix}0 & I\\ -K_p & -K_d\end{bmatrix}\begin{bmatrix}e\\ \dot{e}\end{bmatrix} \implies \dot{x} = Ax$$
- if the real parts of the eigenvalues of A are less than 0 then that implies $e(t) \to 0, t \to \infty$, this is **Feedback Linearizing Control**
- if K_d and K_p are diagonal matrices then they need to be positive definite i.e. $z^T Mz > 0$ for all real-valued vectors z, aka **the eigenvals of K_p, K_d are all positive**
- when given $\ddot{x} + b\dot{x} + c = 0$ the characteristic eqn. is $\lambda^2 + b\lambda + c = 0, x(t) = c_1 e^{\lambda_1 t} + c_2 e^{\lambda_2 t}$
- **Gravity Compensation Control:** want to hold an obj steadily against gravity, if $\dot{\theta} \tilde{=} 0 \implies \Gamma = N(\theta)$
- **Gravity Compensation + PD:** $\Gamma = N(\theta) - K_p e - K_d\dot{e}$
- **PID:** ***proportional:*** does most of the work to pull state to desired traj, ***derivative:*** dampens proportional, prevents oscillation and overcorrection, allows for convergence, ***integral:*** corrects steady-state error caused by constant forces like g, can be thought as supplying force to keep error at 0
- **Model Based Control:** $u = u_{ff} + u_{fb}$