ME 449: Robotic Manipulation Capstone Project

Hannah Huang

This project's task is to calculate the trajectory that the KUKA youBot must take to pick up the block and place it at the goal location (see Figure 1)

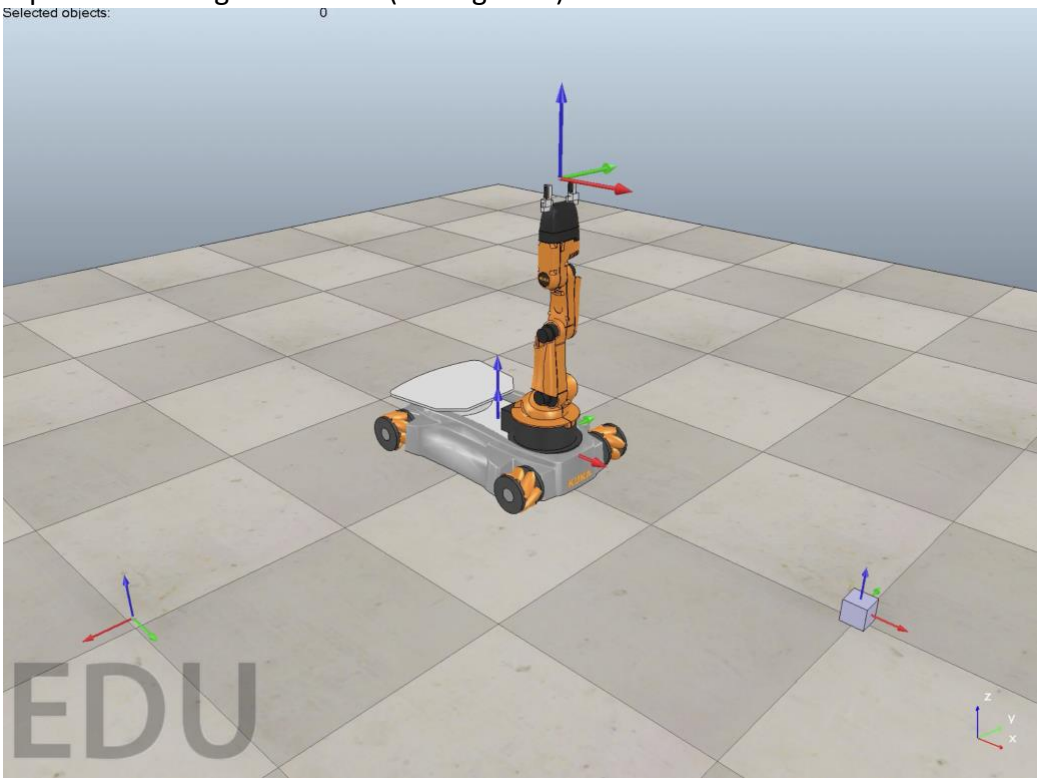

Figure 1: Initial conditions of Kuka youBot, with block in bottom right and goal location in bottom left.

The calculations are all performed in MATLAB (see Appendix 1) and simulated in CoppeliaSim. I will demonstrate my best-case scenario, along with an example of overshooting.

1. "youBotManipulator.m"

This takes inputs of

- *Tsc_initial*: Initial cube location in {s} frame.
- *Tsc_final*: Final cube location in {s} frame.
- *Tse_initial*: Initial end-effector location in {s} frame.
- *config*: Initial youBot configuration (12-vector of for chassis config, joint angles, and wheel angles).
- *Kp*: Proportional gain matrix.
- *Ki*: Integral gain matrix.

It outputs:

- *configs*: matrix of configurations in 13-vector format
- *errors*: matrix of Xerr
- *timesteps*: array of times (for graphing Xerr)

The max joint/wheel speeds are set to 12 rad/s. After declaring all the constants of the youBot (such as information about the wheel and chassis configurations, the joint screw axes, etc.), the ideal trajectory of the end-effector is computed with TrajectoryGenerator.m (see section 2). Then, the configurations of the computed trajectory are looped through, starting from the first configuration to the second-to-last configuration. This is because on the i-th loop, we set the desired configuration to the i-th computer configuration, and then we set the *next* desired configuration to the i+1-th configuration. Using the known current configuration, as well as the desired configurations, the twist and error are computed with FeedbackControl.m (see section 3). Then, I calculate the combined arm and joint Jacobians

$$J_e(\theta) = [J_{base}(\theta) \;\; J_{arm}(\theta)] \; .$$

$J_{arm}(\theta)$ is just equal to the body Jacobian (Jacobian of the joints in the {b} frame). $J_{base}(\theta) = \left[\mathrm{Ad}_{T_{0e}^{-1}(\theta)T_{b0}^{-1}}\right] F_6$, where $T_{b0}(\theta)$ is the constant offset between the chassis center and body frame of the arm joints in the {b} frame, and $T_{0e}(\theta)$ is the offset between the end-effector base and end-effector in the {0} frame. $T_{0e}(\theta)$ is recalculated every loop, just by using FKinBody.m, since we know M_{0e} (home configuration of end-effector in the {0} frame), the body screw axes, and an updated list of joint axes. Next, wheel speeds can be calculated knowing that

$$\begin{bmatrix} u \\ \dot{\theta} \end{bmatrix} = J_e^{\dagger}(\theta)\mathcal{V} \; .$$

The configuration is then updated using these new wheel speeds using NextState.m (see section 4).

2. "TrajectoryGenerator.m"

Trajectory generator takes the following inputs:

- *Tse_initial*: The initial configuration of the end-effector in the reference trajectory.
- *Tsc_initial*: The cube's initial configuration.
- *Tsc_final*: The cube's desired final configuration.
- *Tce_grasp*: The end-effector's configuration relative to the cube when it is grasping the cube.
- *Tce_standoff*: The end-effector's standoff configuration above the cube, before and after grasping, relative to the cube.
- *k*: The number of trajectory reference configurations per 0.01 seconds.

And has the following output:

- *trajectory*: A representation of the N configurations of the end-effector along the entire concatenated eight-segment reference trajectory

The motion of the end-effector is broken up into 8 trajectory segments, as shown in Table 1. The following configurations are defined as follows:

*Tse_initialstandoff = Tsc_initial * Tce_standoff*
*Tse_finalstandoff = Tsc_final * Tce_standoff*
*Tse_initialgrasp = Tsc_initial * Tce_grasp*
*Tse_finalgrasp = Tsc_final * Tce_grasp*

Tse_initialstandoff and *Tse_finalstandoff* refer to the configurations when the end-effector is directly above the cube at its initial and final location, respectively. *Tse_initialgrasp* and *Tse_finalgrasp* refer to the configuration when the end-effector is grabbing the cube at its initial and final location, respectively. Segment 1 was given the longest time because it gives the robot more time to correct its path as it moves to grab the cube from its initial starting position.

Table 1. End-effector parameters for each of the 8 trajectory segments. "Xstart" and "Xend" refer to the starting configuration and ending configurations of the individual trajectories. "State" refers to the state of the gripper: 0 for open and 1 for closed. "Time" refers to how long it takes to complete this trajectory.

#	End-effector action description	Xstart	Xend	State	Time (s)
1	Move from initial to above block	*Tse_initial*	*Tse_initialstandoff*	0	10
2	Lower to grab block	*Tse_initialstandoff*	*Tse_initialgrasp*	0	1
3	Close gripper	*Tse_initialgrasp*	*Tse_initialgrasp*	1	0.7
4	Lift block	*Tse_initialgrasp*	*Tse_initialstandoff*	1	1
5	Move block above final position	*Tse_initialstandoff*	*Tse_finalstandoff*	1	6
6	Put block down	*Tse_finalstandoff*	*Tse_finalgrasp*	1	1
7	Open gripper	*Tse_finalgrasp*	*Tse_finalgrasp*	0	0.7
8	Move back above block	*Tse_finalgrasp*	*Tse_finalstandoff*	0	1

For each of the 8 segments, *N* (the number of steps needed to complete a trajectory) was calculated as $N = k * T$, where *T* is the length in seconds of each segment (right-most column of Table 1). Then, the trajectory for each segment is calculated using the ScrewTrajectory function with all the parameters previously defined and *method* = 3 (third-order polynomial time scaling). This returns a cell array of *N* configurations, which make up the trajectory of that segment. Each of these configurations are flattened into the 13-element array, in which the first 12 elements are values from the configuration matrix, and the last element is the state of the gripper as defined in Table 1. The flattened trajectories for each segment are vertically stacked into an array that describes the trajectory of the end-effector over the entire duration of the movement.

3. FeedbackControl.m

This take as input:

- *X*: Tse,real,k
- *Xd*: Tse,desired,k
- *Xdnext*: Tse,desired,k+1
- *Kp*: Proportional gains
- *Ki*: Integral gains
- *dt*: Timestep
- *cumXerr*: Cumulative Xerr

Output:

- *V*: Twist
- *Xerr*: Error twist

This code is very simple, and it just implements the feed-forward PI controller:

$$\mathcal{V}(t) = [\mathrm{Ad}_{X^{-1}X_d}]\mathcal{V}_d(t) + K_p X_{err}(t) + K_i \int_0^t X_{err}(t)dt \ .$$

To implement the integral controller, I had to add "cumXerr" to the inputs of this program (not included in the original instructions of milestone 3). In youBotManipulator.m, the error twist is added cumulative error on every loop.

4. NextState.m

This takes as input:

- *config*: A 1x12 vector representing the current configuration of the robot (3 variables for the chassis configuration, 5 for the arm configuration, and 4 variables for the wheel angles).
- *speeds*: A 1x9 vector of controls indicating the wheel speeds u (4 variables) and the arm joint speeds dtheta (5 variables).
- *dt*: A timestep.
- *F*: Inverse of the H matrix
- *Tsb:* Body frame relative to space frame
- *maxspeed:* Max and min angular speeds.

Outputs:

- *thetas_new*: A 6x1 vector representing joint angles at the next timestep.
- *wheel_angs_new*: A 4x1 vector representing wheel angles at the next timestep.
- *q_new*: A 3x1 vector representing chassis configuration at the next timestep.

The $F(\theta)$ and $Tsb(\theta)$ are already calculated in youBotManipulator.m because these are important for calculating wheel speeds and updating the current configuration, so these are inputs into NextState.m, instead of calculating them again inside of NextState.m. This code uses first-order Euler-step to update the arm joint angles and wheel angles. Then, we use odometry to update the chassis configuration. First we find the twist

$$\mathcal{V}_b = F(u * \Delta t) \ ,$$

where we multiply u (the wheel speeds) by the change in time over which u was calculated because we want the change in wheel position for a change in time equal to 1s. We convert $\mathcal{V}_b \rightarrow \mathcal{V}_{b6}$ and then we compute

$$T_{sb_{k+1}} = T_{sb_k} e^{[\mathcal{V}_{b6}]} \ .$$

And now we have all the information of the updated chassis configuration.

5. Results

In my "best" case, I show very little overshoot, as I do not include an integral gain (see Figure 2). In my "overshoot" case, I add a relatively large integral gain, and the result shows large overshooting (see Figure 3). Both cases show a large increase in the error plots starting ~14.6s, and peaking ~16s. This is during the time that the robot is grasping the cube and has begun moving it to the goal location. This increase in error is due to a rapidly changing trajectory that has gone from linear motion to a more complicated rotational motion. In the new task case, a similar large increase in error is seen near the beginning, around 6.7s. This is just after the robot initially straightens itself out to get the end-effector around the initial position. This increase in

errors is caused by the trajectory rapidly from an end-effector that is facing in the +x direction, to an end-effector that is rapidly twisting to face in the -y direction.

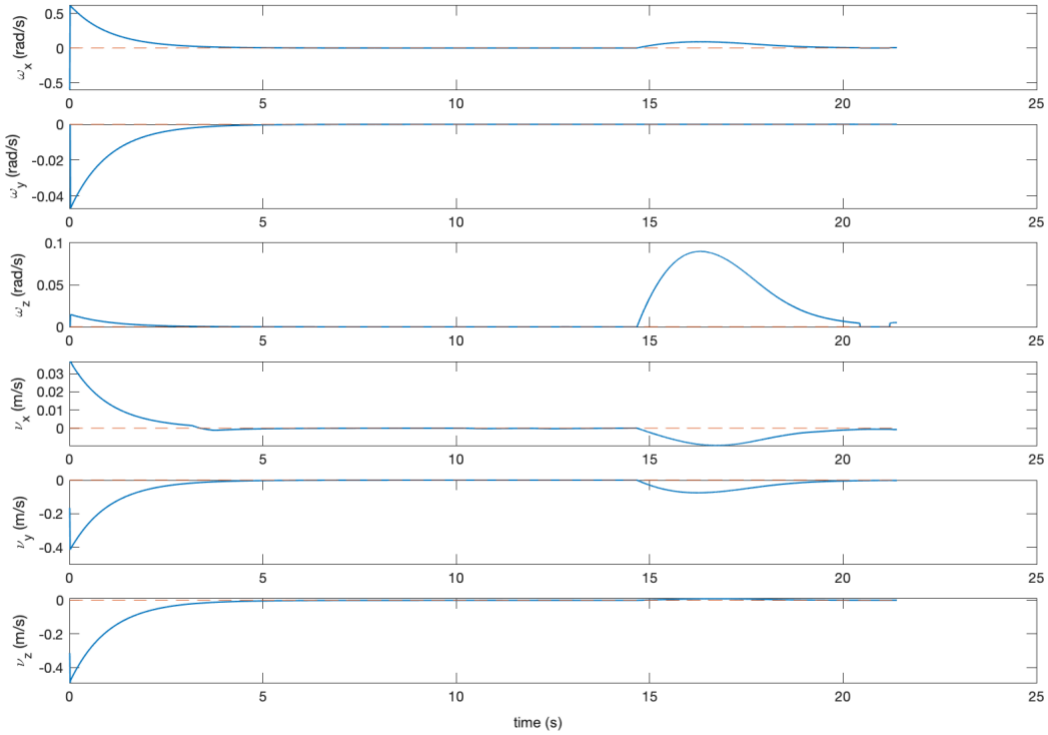

Figure 2: Angular and linear velocities over time for "best" response

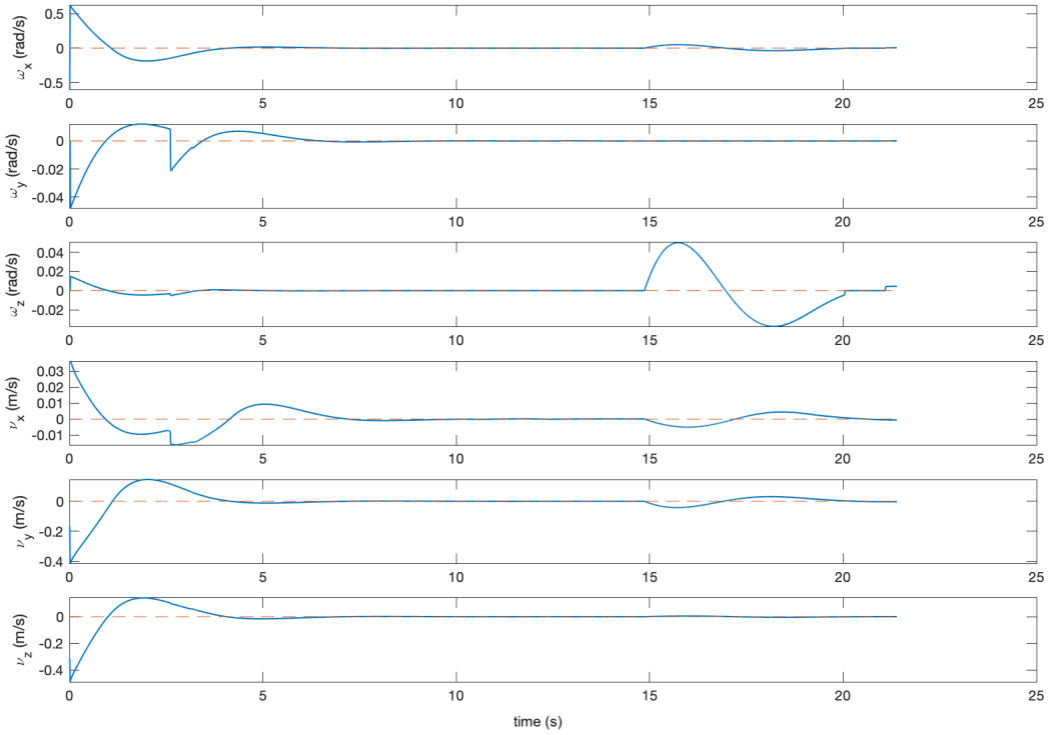

Figure 3: Angular and linear velocities over time for "overshoot" response

To tune all of these controllers, I had to make sure that the error settled by the time the arm went to grab the cube, and then again, when the arm set down the cube. Making sure the error settled by the time we grab the cube was more difficult, since if we achieve the first situation, then the latter situation will start off really close to the desired position. Tuning the overshoot case was a bit difficult. I had to find a case where there was enough a large wind-up integral overshoot, but not too large that the error wouldn't settle by the time the arm grabbed the cube. First, I increased the length of the first trajectory segment in TrajectoryGenerator.m to give the robot more time to correct itself before reaching to grab the cube. Then, I based this controller off of my best controller, and then adjusted the integral gain until I saw overshooting. At this integral gain, the error hadn't settled by the time the arm reached the cube, so I had to slowly increment both PI gains until I found my desired response.

Appendix A

MATLAB Code

youBotManipulator.m

```
function [configs,errors,timesteps] = youBotManipulator(Tsc_initial, Tsc_final,
Tse_initial, config, Kp, Ki)
% INPUTS:
% Tsc_initial       Initial cube location in {s} frame.
% Tsc_final         Final cube location in {s} frame.
% Tse_initial       Initial end-effector location in {s} frame.
% config            Initial YouBot configuration (12-vector of for chassis
%                   config, joint angles, and wheel angles).
% Kp                Proportional gain matrix.
% Ki                Integral gain matrix.
% OUTPUTS:
% configs           matrix of configurations in 13-vector format
% errors            matrix of Xerr
% timesteps         array of times (for graphing Xerr)

% change this value to change joint speed limits
maxspeed = 12;

% timestep should be 10ms to match CoppeliaSim
dt = 0.01;

% joint constants
Blist = [0,0,1,0,0.033,0;
         0,-1,0,-0.5076,0,0;
         0,-1,0,-0.3526,0,0;
         0,-1,0,-0.2176,0,0;
         0,0,1,0,0,0]';

% cube-end effector configurations
Tce_standoff = [-0.7071,0,0.7071,0.015; 0,1,0,0; -0.7071,0,-0.7071,0.15; 0,0,0,1];
Tce_grasp = [-0.7071,0,0.7071,0.015; 0,1,0,0; -0.7071,0.015,-0.7071,0; 0,0,0,1];

% wheel/chassis initializations
M0e = [1 0 0 0.033; 0 1 0 0; 0 0 1 0.6546; 0 0 0 1];
Tb0 = [1 0 0 0.1662; 0 1 0 0; 0 0 1 0.0026; 0 0 0 1];
r = 0.0475;
gammas = [-pi/4, pi/4, -pi/4, pi/4];
xs = [0.47/2, 0.47/2, -0.47/2,-0.47/2];
ys = [0.3/2, -0.3/2, -0.3/2, 0.3/2];
betas = [0,0,0,0];
phi = config(1); x = config(2); y = config(3); z = 0.0963;
Tsb = [cos(phi), -sin(phi), 0, x;
       sin(phi),  cos(phi), 0, y;
              0,         0, 1, z;
              0,         0, 0, 1];

% calulate H and F matrix for Odometry
H = zeros(4,3);
for ii = 1:size(H,1)
    H(ii,:) = (1/r) * [1, tan(gammas(ii))] * ...
        [cos(betas(ii)), sin(betas(ii)); -sin(betas(ii)), cos(betas(ii))] * ...
        [-ys(ii), 1, 0; xs(ii), 0, 1];
end
F = pinv(H);
F6 = [0 0 0 0;0 0 0 0;F;0 0 0 0];

% initialize first configuration matrix (X = Tse)
```

```
thetalist = config(4:8)';
T0e = FKinBody(M0e,Blist,thetalist);
X = Tsb*Tb0*T0e;
% cumulative Xerr starts at 0
cumXerr = zeros(6,1);

fprintf('Generating trajectory...\n')
% generate trajectory
Traj_flat = TrajectoryGenerator(Tse_initial, Tsc_initial, Tsc_final, Tce_grasp,
Tce_standoff, 1);
N = size(Traj_flat,1);

fprintf('Calculating Path...\n')
configs = zeros(N-1,13);
errors = zeros(N-1,6);
for k=1:N-1
    % calculate twist and error
    [Xd,gripper] = unflatten_trans(Traj_flat(k,:)); % turn 13-vec into 4x4 and gripper
state
    [Xdnext,~] = unflatten_trans(Traj_flat(k+1,:));
    [V,Xerr] = FeedbackControl(X,Xd,Xdnext,Kp,Ki,dt,cumXerr);
    cumXerr = cumXerr + Xerr * dt;
    errors(k,:) = Xerr';
    % calculte Je and speeds
    Jarm = JacobianBody(Blist,thetalist);
    Jbase = Adjoint(T0e\inv(Tb0))*F6;
    Je = [Jbase Jarm];
    speeds = (pinv(Je,1e-4)*V)';
    % calculate the next configuration
    [thetalist, wheel_angs, q] = NextState(config, speeds, dt, F, Tsb, maxspeed);
    config = [q', thetalist', wheel_angs'];
    % use the new config to update values
    T0e = FKinBody(M0e,Blist,thetalist);
    phi = config(1); x = config(2); y = config(3); z = 0.0963;
    Tsb = [cos(phi), -sin(phi), 0, x;
           sin(phi),  cos(phi), 0, y;
                  0,         0, 1, z;
                  0,         0, 0, 1];
    X = Tsb*Tb0*T0e;

    % write configs to csv matrix
    configs(k,:) = [config,gripper];
end

timesteps = (1:N-1)*dt;

function [T,gripper] = unflatten_trans(traj_flat)
% unflattens the 13-vector configuration into a transformation matrix and
% gripper state
T = [traj_flat(1:3),traj_flat(10); traj_flat(4:6),traj_flat(11);
     traj_flat(7:9),traj_flat(12); 0 0 0 1];
gripper = traj_flat(end);
end

end

function [T,gripper] = unflatten_trans(traj_flat)
% unflattens the 13-vector configuration into a transformation matrix and
% gripper state
T = [traj_flat(1:3),traj_flat(10); traj_flat(4:6),traj_flat(11);
     traj_flat(7:9),traj_flat(12); 0 0 0 1];
gripper = traj_flat(end);
end
```

TrajectoryGenerator.m

```
function Traj_flat = TrajectoryGenerator(Tse_initial, Tsc_initial, Tsc_final,
Tce_grasp, Tce_standoff, k)
% % To create the .csv for Milestone 2, copy and paste the following code:
% >> milestone2.m
% trajectory = TrajectoryGenerator(Tse_initial, Tsc_initial, Tsc_final, Tce_grasp,
Tce_standoff, k)
% Inputs:
%   Tse_initial     The initial configuration of the end-effector in the
%                   reference trajectory.
%   Tsc_initial     The cube's initial configuration.
%   Tsc_final       The cube's desired final configuration.
%   Tce_grasp       The end-effector's configuration relative to the cube
%                   when it is grasping the cube.
%   Tce_standoff    The end-effector's standoff configuration above the
%                   cube, before and after grasping, relative to the cube.
%   k               The number of trajectory reference configurations per
%                   0.01 seconds.
% Outputs:
%   trajectory      A representation of the N configurations of the
%                   end-effector along the entire concatenated
%                   eight-segment reference trajectory.

% Trajectory is broken up into 8 segments:
%   1. From initial to location above block:
%       Tse_initial --> Tse_initialstandoff
%   2. Lower to grab block:
%       Tse_initialstandoff --> Tse_initialgrasp
%   3. Close gripper
%   4. Lift block:
%       Tse_initialgrasp --> Tse_initialstandoff
%   5. Move block above final position:
%       Tse_initialstandoff --> Tse_finalstandoff
%   6. Put block down:
%       Tse_finalstandoff --> Tse_finalgrasp
%   7. Open gripper
%   8. Move end-effector back above block:
%       Tse_finalgrasp --> Tse_finalstandoff

Tse_initialstandoff = Tsc_initial * Tce_standoff;
Tse_initialgrasp = Tsc_initial * Tce_grasp;
Tse_finalstandoff = Tsc_final * Tce_standoff;
Tse_finalgrasp = Tsc_final * Tce_grasp;

% rows correspond to segment
% total time taken for each trajectory segment
T = [10, 1, 0.7, 1, 6, 1, 0.7, 1]';
% number of steps taken for each trajectory segment
N = k*T/0.01;
% state of gripper (0 opened, 1 closed)
gripper = [0,0,1,1,1,1,0,0];
% starting/ending configurations
Xstart = {Tse_initial,Tse_initialstandoff,Tse_initialgrasp,...
    Tse_initialgrasp,Tse_initialstandoff,Tse_finalstandoff,...
    Tse_finalgrasp,Tse_finalgrasp};
Xend = {Tse_initialstandoff,Tse_initialgrasp,Tse_initialgrasp,...
    Tse_initialstandoff,Tse_finalstandoff,Tse_finalgrasp,...
    Tse_finalgrasp,Tse_finalstandoff};
% initiate flat trajectory matrix
Traj_flat = zeros(sum(N),13);
idx = 0;

% fill array
```

```
for jj = 1:8
    traj = ScrewTrajectory(Xstart{jj}, Xend{jj}, T(jj), N(jj), 3);
    for ll = 1:N(jj)
        idx = idx + 1;
        Traj_flat(idx,:) = flatten_T(traj{ll},gripper(jj));
    end
end

end

function Tflat = flatten_T(T,gripper)
% turns configuration matrix and gripper status to the 13-valued array
Tflat = [T(1,1:3), T(2,1:3), T(3,1:3), T(1,4), T(2,4), T(3,4), gripper];
end
```

FeedbackControl.m

```
function [V,Xerr] = FeedbackControl(X,Xd,Xdnext,Kp,Ki,dt,cumXerr)
% Input:
% X         Tse,real,k
% Xd        Tse,desired,k
% Xdnext    Tse,desired,k+1
% Kp        Proportional gains
% Ki        Integral gains
% dt        Timestep
% cumXerr   Cumulative Xerr
% Output:
% V         Twist
% Xerr      Error twist
Vd = se3ToVec((1/dt) * MatrixLog6(Xd\Xdnext));
Xerr = se3ToVec(MatrixLog6(X\Xd));
V = Adjoint(X\Xd)*Vd + Kp*Xerr + Ki*cumXerr;

end
```

NextState.m

```
function [thetas_new, wheel_angs_new, q_new]=NextState(config, speeds, dt, F, Tsb,
maxspeed)
% Input:
% config        A 1x12 vector representing the current configuration of the
%               robot (3 variables for the chassis configuration, 5
%               variables for the arm configuration, and 4 variables
%               for the wheel angles).
% speeds        A 1x9 vector of controls indicating the wheel speeds u (4 variables)
%               and the arm joint speeds dtheta (5 variables).
% dt            A timestep.
% F             Inverse of the H matrix
% Tsb           Body frame relative to space frame
% maxspeed      Max and min angular speeds.
% Output:
% thetas_new        A 6x1 vector representing joint angles at the next timestep.
% wheel_angs_new    A 4x1 vector representing wheel angles at the next timestep.
% q_new             A 3x1 vector representing chassis configuration at the next
timestep.

% configurations
thetas = config(4:8)';
wheel_angs = config(9:end)';
% bound angular speeds
speeds(speeds>maxspeed) = maxspeed; speeds(speeds<-maxspeed) = -maxspeed;
% initialize speeds
u = speeds(1:4)';
dthetas = speeds(5:end)';
```

```
% First-order Euler step for new joint and wheel angles
thetas_new = thetas + dthetas*dt;
wheel_angs_new = wheel_angs + u*dt;

Vb = F * u*dt;
Vb6 = [0; 0; Vb; 0];
Tbb_new = MatrixExp6(VecTose3(Vb6));
Tsb_new = Tsb*Tbb_new;

q_new = [acos(Tsb_new(1,1)), Tsb_new(1,4), Tsb_new(2,4)]';

end
```