KTH Royal Institute of Technology

EL2700 - Model Predictive Control

Assignment 4: Model Predictive Control

Division of Decision and Control Systems
School of Electrical Engineering and Computer Science

Authors:
Jacopo Dallafior,Stefano Tonini

September 22, 2024

Contents

Chapter 1

Design Task

Introduction

In this report, we address the tasks and questions presented in Assignment 4. The focus is on designing and analyzing a linear Model Predictive Controller (MPC) for the Astrobee. This includes computing control invariant sets, evaluating the influence of control horizons and constraints, and simulating the Astrobee with different setups.

1.1 Part I - The MPC Setup

Q1: Study the Influence of the Control Horizon

We computed the control invariant set $K_N(X_N)$ for control horizons $N = 5$, $N = 10$, and $N = 20$, while keeping the state and control constraints fixed.
The control horizon N in Model Predictive Control (MPC) refers to the number of steps ahead that the controller considers when making decisions. Larger values of N allow the controller to optimize over a longer prediction horizon, which typically leads to more stable and robust control actions. However, increasing N also increases computational complexity.

The following figures show the invariant sets for different control horizons:

Invariant Set for N=5: The first plot shows the invariant set when the control horizon is 5 steps. The set is relatively small, meaning that the system has fewer states it can reach while satisfying the constraints within 5 steps. This results in limited flexibility in control actions.

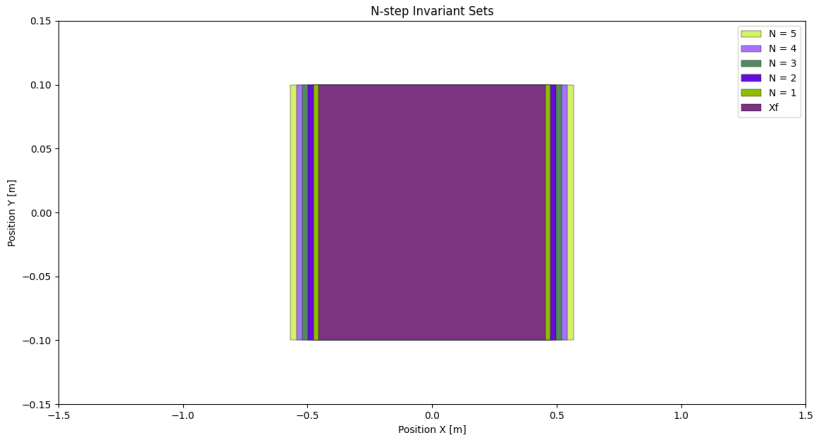


Figure 1.1: Invariant set for $N = 5$

Invariant Set for N=10 : In the second plot for N=10, the size of the invariant set has increased compared to N=5. This indicates that, with a longer prediction horizon, the controller can plan further ahead, allowing the system to reach a larger set of states while still respecting the control and state constraints.

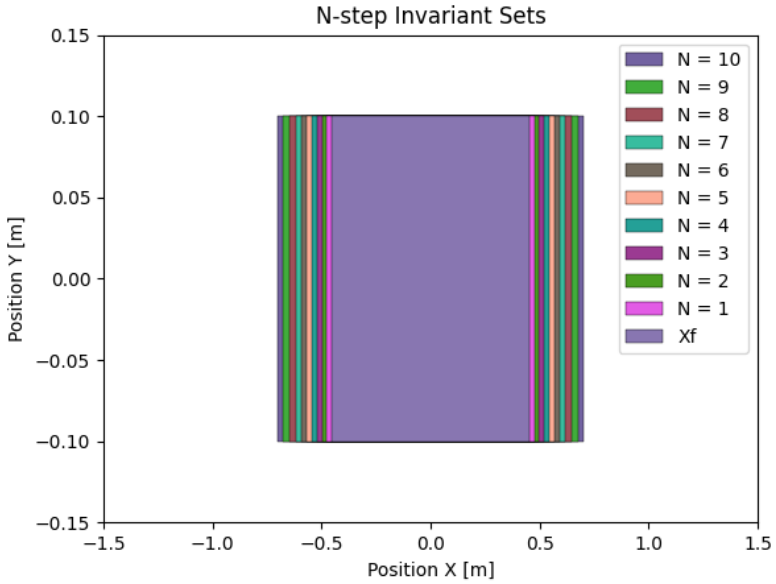


Figure 1.2: Invariant set for $N = 10$

Invariant Set for N=20: In the final plot for N=20, the invariant set is even larger. This demonstrates that the system's ability to reach more states increases as N grows, which enhances the overall feasibility of the control problem.

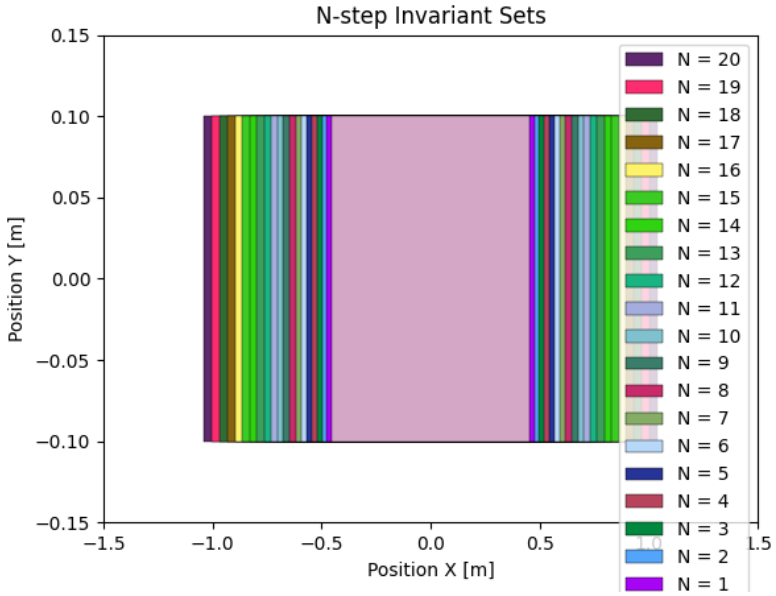


Figure 1.3: Invariant set for $N = 20$

For small N: The control is less flexible, and the system may struggle to respect the constraints in a short horizon. For larger N: The control becomes more robust, with a larger set of states that can be safely controlled, but the computational demand increases.

Q2: Influence of Control Constraints

Control constraints limit the authority of the controller over the system. In the MPC setup, the control input u is restricted by bounds $u_{\min} \leq u_t \leq u_{\max}$. When these bounds are relaxed (i.e., increased), the controller has more flexibility, which typically leads to a larger controllable set. In this question, we study the influence of relaxing the control constraints by multiplying the original control bounds $u_{\max}$ by a factor of 3.

In the plot below, we visualize the control invariant sets for two scenarios:

- **Original Control Bounds** $|u|$ (in green)
- **Relaxed Control Bounds** $3|u|$ (in gray)

The green region represents the controllable set with the original control constraints, and the gray region represents the expanded controllable set when the control limits are multiplied by 3. As expected, when the control limits are increased, the system can reach a significantly larger set of states while still satisfying the constraints.

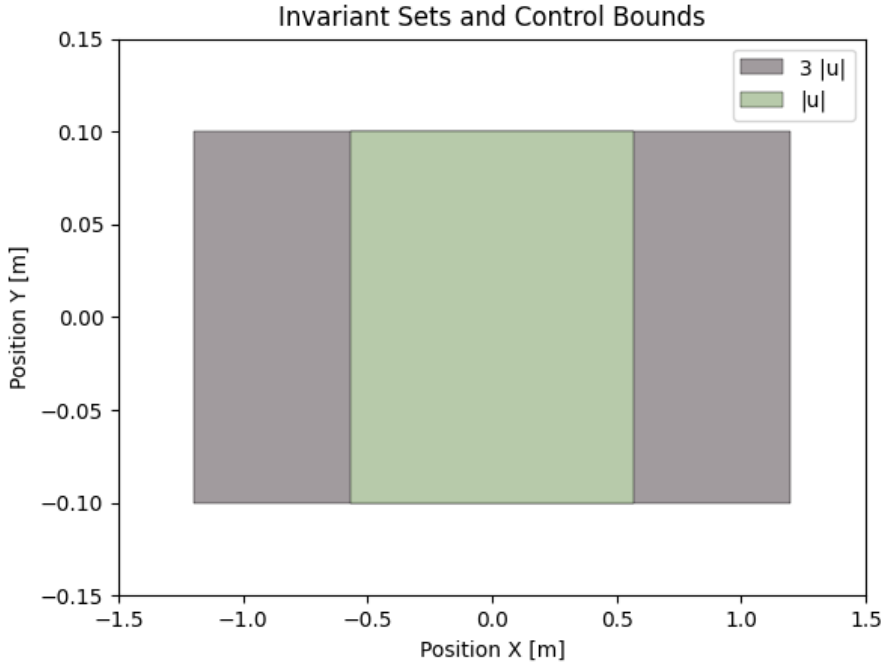


Figure 1.4: Control Invariant Sets for $|u|$ and $3|u|$

Increasing the control constraints allows the system to operate over a larger state space, making the controller more flexible. However, this comes at the potential cost of increased energy consumption or actuator usage, as the controller may apply higher forces or torques.

1.2 Part II - The MPC Implementation

Q3: Analysis of MPC Implementation

Q3: Examine the MPC Controller Implementation

In this section, we will examine the Model Predictive Controller (MPC) implementation in the file *mpc.py*. We will address the following points:

1. How the state constraints are set;
2. How the objective function is formulated;
3. How the terminal constraint is set;
4. What the variable `param_s` holds;
5. Why only the first element of the control prediction horizon is selected in the `mpc_controller` method.

The first thing done in the code is the initialization of the cost function weights, where the matrices are setted in this way:

```
if P is None:
    P = np.eye(self.Nx) * 10
if Q is None:
    Q = np.eye(self.Nx)
if R is None:
    R = np.eye(self.Nu) * 0.01
```

3.1 State Constraints Setting

The state constraints are set using upper and lower bounds for both the state variables and control inputs, as shown in the following code:

```
    if xub is None:
        xub = np.full((self.Nx), np.inf)
    if xlb is None:
        xlb = np.full((self.Nx), -np.inf)
    if uub is None:
        uub = np.full((self.Nu), np.inf)
    if ulb is None:
        ulb = np.full((self.Nu), -np.inf)
```

Lines 83-90 check if the upper and lower bounds for states (`xub`, `xlb`) and control inputs (`uub`, `ulb`) are provided. If they are not, they are initialized to positive infinity (`np.inf`) or negative infinity (`-np.inf`), respectively, which implies no constraints for upper and lower bound.

The code then check if it is provided an upper and lower bound with an *if* condition, then it checks which element of the bound is finite. For those elements it adds an inequality constraint, to ensure the state values do not exceed the bounds. This is done for the upper and lower bound of the state as can be seen from the code that follows.

```
if xub is not None:
    non_inf_indx = (xub != np.inf).flatten()
    if np.any(non_inf_indx):
        con_ineq     += [x_t[non_inf_indx] <= xub[non_inf_indx].flatten
           ()]
if xlb is not None:
    non_inf_indx = (xlb != - np.inf).flatten()
    if np.any(non_inf_indx):
        con_ineq     += [x_t[non_inf_indx] >= xlb[non_inf_indx].flatten
           ()]
```

For each time step `t`, the state constraints are set by specifying the upper (`con_ineq_ub`) and lower bounds (`con_ineq_lb`) for the state vector `x_t`.

3.2 Objective Function Formulation

The objective function, which the MPC tries to minimize, is formulated as:

$$J = \sum_{k=0}^{N-1} x_k^T Q x_k + u_k^T R u_k + x_N^T P x_N$$

where Q, R, and P are weighted matrices for the states, control inputs, and terminal states.
We implement this function using self.running_cost and self.terminal_cost:

```
# Objective Function / Cost Function
obj += self.running_cost((x_t - x0_ref), self.Q, u_t, self.R)
# Terminal Cost
obj += self.terminal_cost(x_vars[:, self.Nt] - x0_ref, self.P)
```

The function are defined it the line 194 using the function *quad_form*:

```
    self.running_cost = lambda x, Q, u, R: cp.quad_form(x,Q) + cp.
        quad_form(u,R)
    self.terminal_cost = lambda x, P: cp.quad_form(x,P)
```

self.running_cost: This function represents the running cost at each time step, which is combination of the quadratic forms of the state deviation and the control effort, represented as:

$$x^T Q x + u^T R u \tag{1.1}$$

self.terminal_cost: This represents the cost at the final state of the prediction horizon, to ensure the system state reaches a desired target, represented as:

$$x^T P x \tag{1.2}$$

3.3 Terminal Constraint Setting

The terminal constraint is set as follows:

```
# Terminal constraint
if terminal_constraint is not None:
    # Should be a polytope
    H_N = terminal_constraint.A
    if H_N.shape[1] != self.Nx:
        print("Terminal constraint with invalid dimensions.")
        exit()
    H_b = terminal_constraint.b
    con_ineq += [H_N @ x_vars[:,self.Nt] <= H_b]
```

Here, the terminal constraint uses a polytope defined by the matrices `H_N` and `H_b`. The constraint ensures that with an inequality *con_ineq* the terminal state lies within a specific region. To ensure that a terminal constraint is used at the beginning we use an *if* condition.

3.4 Variable `param_s`

The variable `param_s` is not defined in our code. However in the code the state parameters are defined as:

```
        x0 = cp.Parameter(self.Nx)
        u0 = cp.Parameter(self.Nu)
        x0_ref = cp.Parameter(self.Nx)
```

3.5 First Element of the Control Prediction Horizon

MPC optimizes the control input over a finite prediction horizon. This results in a squence of control actions { u_t, u_t+1, u_t+2, . . . , u_t+N } at each time step t. We apply to the system only the first contol action u_t , and so after applying this input the prediction horizon shifts forward by one step, and then the optimization problem is re-solved using the updated state information at t+1. This is the rceding-horizon strategy.

If we applied the entire sequence of control actions calculated at the initial time step, the controller would not be able to react to any discrepancies or unexpected events during the prediction horizon. By re-computing the control action at each step, the controller can take into account the latest state information, providing robustness to changes in the system dynamics or environment.

1.3 Part III - MPC for Stabilization

Q4: Simulation Results with Modified Control Bounds

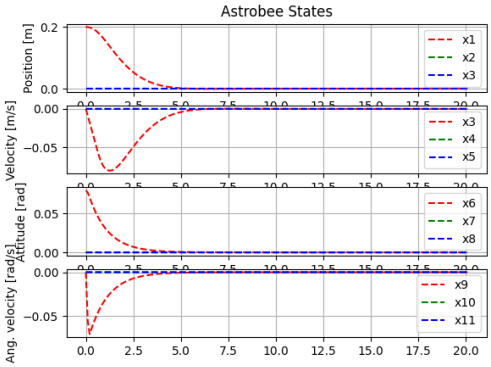
In this section the simulations were conducted using two different sets of control bounds: the original control limits and relaxed bounds, where the limits were increased by a factor of three. The table below summarizes the key results from both simulations:

Simulation Data:

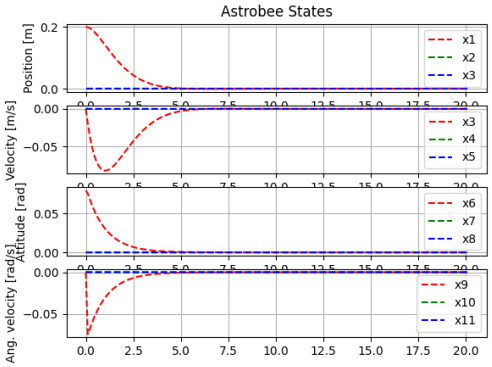
Metric	**Original Bounds**	**Relaxed Bounds**
Energy Used (units)	156.20	160.54
Position Integral Error	3.94	3.56
Attitude Integral Error	0.91	0.89

Table 1.1: Simulation Results: Original vs. Relaxed Control Bounds

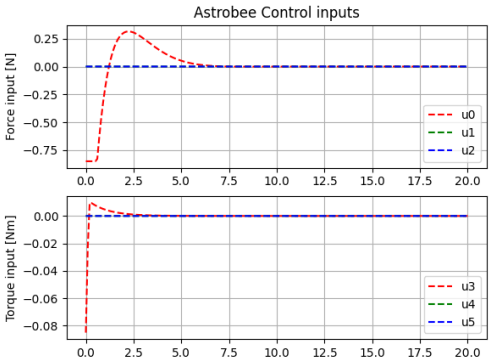
The original control bounds required less energy (156.20 units) compared to the relaxed bounds (160.54 units). However, the relaxed bounds resulted in better tracking performance, with the position and attitude integral errors being slightly lower. The plots on the left show the system states and control inputs with original bounds, where the system stabilizes but with higher error. The right side shows faster convergence of the states and slightly more aggressive control inputs, leading to improved accuracy at the expense of higher energy consumption.



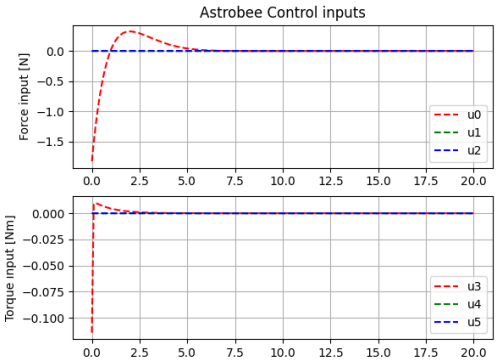
(a) Astrobee States - Original Bounds



(b) Astrobee States - Relaxed Bounds



(c) Astrobee Control Inputs - Original Bounds



(d) Astrobee Control Inputs - Relaxed Bounds

Figure 1.5: Comparison of Astrobee States and Control Inputs for Original and Relaxed Bounds

Q5: Terminal Set Comparison

In this section, we compare the performance of the MPC using two different terminal sets: (i) $X_f = \{0\}$ and (ii) $X_f = X_f^{\text{LQR}}$. The objective is to understand the impact of the terminal set on the feasibility and performance of the controller.

When $X_f = \{0\}$:

- The simulation failed because the MOSEK solver could not find a feasible solution. This indicates that forcing the system to exactly reach the terminal state $X_f = \{0\}$ was too restrictive, leading to an infeasible optimization problem.

When $X_f = X_f^{\mathbf{LQR}}$:

- The simulation ran successfully with the following results:
  - Energy Used: 156.20 units.
  - Position Integral Error: 3.94.
  - Attitude Integral Error: 0.91.

The comparison between using $X_f = \{0\}$ and $X_f = X_f^{\text{LQR}}$ shows that the terminal set greatly affects the feasibility of the MPC problem. When the terminal set is $X_f = \{0\}$, the problem becomes too restrictive, causing the solver to fail. In contrast, using the LQR terminal set X_f^{LQR} allows for a feasible solution with good tracking performance and acceptable energy usage.

The flexibility of the LQR terminal set ensures that the system can stabilize within the constraints, while the exact-zero terminal condition leads to infeasibility due to overly stringent requirements.

Q6: Impact of Increasing Horizon Length to $N = 50$

In the previous section, we encountered an issue where using a time horizon of $N = 10$ led to an infeasible solution. The solver was unable to find a feasible solution, indicating that the time horizon was too short for the system to reach the terminal set within the constraints.

To address this, we increased the MPC horizon length to $N = 50$, and the problem became solvable. The table below summarizes the key results:

Metric	**Result**
MPC Computation Time (sec)	0.058805
MPC Cost	6.719×10^{-10}
Energy Used (units)	156.23
Position Integral Error	3.94
Attitude Integral Error	0.90

Table 1.2: Simulation Results with Horizon Length $N = 50$

The increase in horizon length resolved the infeasibility issue encountered with $N = 10$. The computation time remained low at 0.059 seconds, indicating that the solver could handle the larger horizon without significant delay. The MPC cost was very low, suggesting an optimal solution. However, energy consumption and tracking errors (position and attitude) remained nearly the same as with shorter horizons, meaning that the extended horizon did not significantly improve these performance metrics. Therefore, a shorter horizon may still be preferable for efficiency, but extending the horizon ensures feasibility in certain cases.

Q7: Tuning Parameters

We investigated the effects of tuning the matrices Q and R:

- **Case 1** Multiplying R by 10:
  Multiplying the R matrix by 10 increases the weight of control efforts. This leads to a reduction in energy usage compared to the initial cases where R was multiplied by 0.01. The position and attitude errors remain relatively low, suggesting that the control system is more conservative and uses less energy, but still maintains a reasonable level of performance. As can be seen in the graph (a) below, in this case we have a smooth and stable behavior, but at the same time a slower converge.

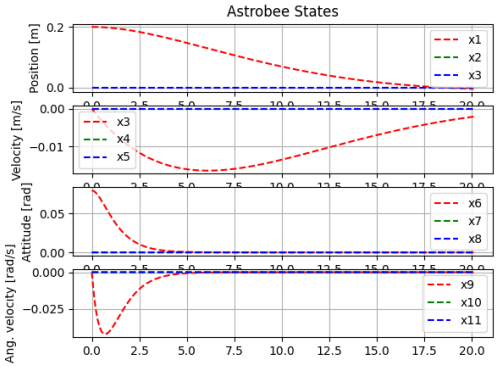
- **Case 2** Adding 100 to Velocity Components of Q and Multiplying R by 10:
  By adding 100 to the velocity components of Q, the system gives more importance to velocity errors. This results in significantly higher attitude error, indicating instability in maintaining the desired orientation. However, the energy usage is the lowest, suggesting that the controller is not overcorrecting for position and velocity errors, possibly leading to underdamped responses. The system is not efficiently maintaining position and orientation, which is evident from the higher integral errors. This tuning cause the system to react slowly to position changes, compromising overall stability.

- **Case 3** Adding 100 to Position and Attitude Components of Q and Multiplying R by 10:
  Increasing the weight of position and attitude components in Q leads to much better performance in terms of lower position and attitude errors. This comes at a cost of significantly higher energy usage, indicating that the controller is working much harder to maintain precise control over position and orientation. It results in good tracking performance, but the initial oscillation in velocity suggest that the tuning might be too aggressive, requiring more energy to control this initial oscillations. The graph (c) show, as expected, a faster convergence for what concern the position and attitude.

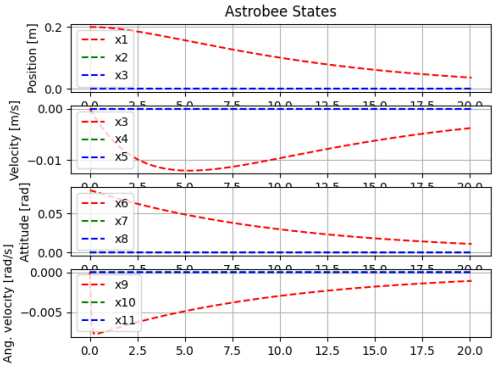
- **Case 4** Increasing All Elements of Q and Multiplying R by 10:
  Increasing all elements of Q by 100 results in low position and attitude errors, indicating a very tightly controlled system. However, this also results in high energy usage, as expected, because the system is aggressively correcting any deviations in all states. This tuning results in aggressive control action, minimizing position and attitude errors effectively.

Case	**Energy used**	**Position integral error**	**Attitude integral error**
Case 0	156.22913060096448	3.941556225319951	0.9022840679879316
Case 1	30.890485701463632	16.843581911353294	1.1657621263240288
Case 2	19.845579190697258	22.082294070430258	7.046479747916053
Case 3	108.37131148400032	5.7056214991625005	0.4571008220813323
Case 4	100.83823715770612	5.80730867799961	0.9057606849641491

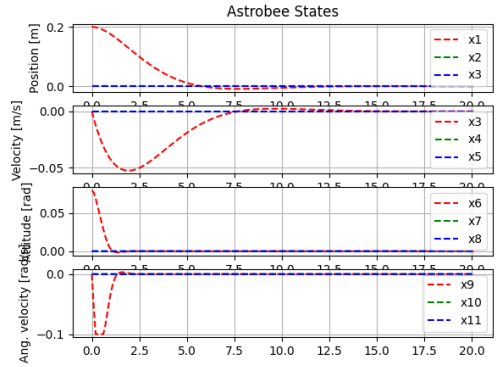
Table 1.3: Simulation results for different cases.



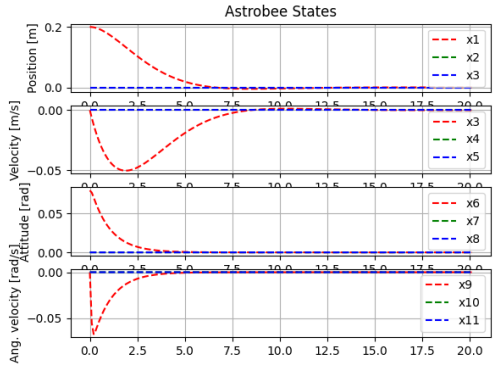
(a) case 1: Multiply R by 10



(b) case 2: Add 100 to the velocity components



(c) case 3: Add 100 to the position and attitude components



(d) case 4: Increase all elements of Q by 100