CS-8803 Special Topic: Machine Learning for Robotics

Homework

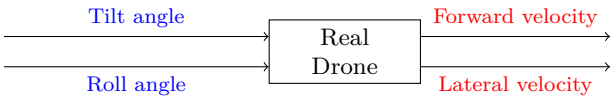
In this homework on system identification, you are going to identify the model of a simulated robot by collecting input-output data, choosing a model structure, and estimating the model parameters with the maximum likelihood method.

You will use V-REP for physical simulation, ROS for interfacing with the simulated environment (controlling the robot and collecting data), and Matlab for identifying the model of the robot (estimating its parameters). You can use Matlab's System Identification toolbox if you want, and in particular its GUI, but it is not necessary.

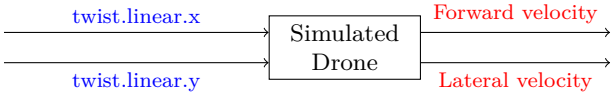
Please submit any Matlab code you wrote and a short report by email by next class (don't forget to mention your group name).

Introduction

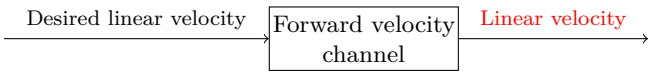
The scene drone is a non-linear MIMO system. We will ignore its attitude and angular control and just focus on its horizontal velocities. Horizontal velocity on a drone is achieved by balancing the force resulting from the drone angle with the air resistance. There are two angles that can be controlled and two horizontal velocities, so it has two inputs to outputs:



A simulated drone is similar, but in addition, there is an internal control law that will adjust the tilt and roll angles to maintain desired velocities, so in practice the input we use is a twist, and this is the system we will try to identify:



It is this system that we are going to try and identify, in physical simulation with V-REP. Since it is still non-linear and MIMO, we will only identify the forward velocity channel. This channel is linear and SISO, with the following input and output:



We will still call this channel the drone, $\mathcal{B}$ for short.

1 Design an experiment and collect data

To identify the drone's linear velocity channel, pictured in the previous block diagram, we need to feed the drone various forward and backward linear velocity commands and record both this input signal and the corresponding output for further analysis.

To publish desired linear velocities to the drone, we will use a gamepad and the teleoperation package you made for the drone two weeks ago: wiki.ros.org/joy/Tutorials.

To record the input and output signals of the linear velocity channel, we will use ROS data recording capability: wiki.ros.org/ROS/Tutorials/Recording and playing back data.

Start up the ROS master and V-REP and load the drone-minimal.ttt scene. Reminders:

- Run `roscore` before `vrep.sh` so that V-REP can export its variables to ROS.

- The drone scene is `drone-minimal.ttt`.
- V-REP is installed in two directories:
  - `/cs-share/pradalier`, symlink to `/cs-share/pradalier/V-REP_PLAYER...`, is a light version that does not permit scene modification; the alias `vrep` points to this version
  - `/cs-share/pradalier/V-REP_PRO_EDU...` is the fully-fledged version: use it at the end of the homework when it will be necessary to change the drone's mass

The desired and actual velocities are provided by the simulator under the topics `/vrep/drone/vel_cmd` and `/vrep/drone/out_vel`, respectively. Twists are generalized velocities: three linear and three angular components, in this order.

Process through the following steps to collect input and output signal data with V-REP and ROS:

1. Adapt your drone's teleoperation node to filter out the angular velocity command, that is to say to be able to teleoperate the drone only forward and backward, not left and right.

   This removes the possibility for accidentally activating the the drone's non-linearities with the gamepad.

2. Also, change this node's publication rate so that it publishes `/vrep/twistCommand` at the same rate than `/vrep/twistStatus` is published (`rostopic hz /vrep/twistStatus`).

   This ensures that when recorded, the input and the output will have the same sample rate, thereby avoiding tedious and questionable resampling work later, before any parameter estimation method can be applied. Alternatively, the matlab `interp2` method can do this resampling very easily.

3. Run the joystick node (`rosrun joy joy_node`) and your teleoperation node. Plot the input `/vrep/drone/vel_cmd/linea` and the output `/vrep/drone/out_vel/twist/linear/x` in real time, using `rqt_plot`.

   This is the data that you are going to record. You should observe how the output is a delayed version of the intput, which just proves that the drone's controller effectively does its job of servoing the actual velocity to the desired velocity. You should be able to have an idea of the value of this delay.

   You might also observe aberrant values of the output when the input is positive. Aberrant values are called outliers and should definitely be removed from the collected data, in a preprocessing step, before applying any parameter estimation method.

4. Use ROS data recording capability to collect the system's input and output in a bagfile. It's better to dump them in the same bagfile rather than in separate bagfiles, for synchronization reasons.

   You should collect at least two data sets, one for identification and one for validation.

   A simple step-like signal is probably a good start. It is possible to modulate the amplitude of the step by not pushing the corresponding axis on the gamepad to its maximum.

5. Convert the binary bagfile to text logfiles, separating the input from the output at the same time, using the following command: `rostopic echo -b bagfile.bag -p /vrep/topic > logfile.csv` (`-b` is for "bag" and `-p` is for "print").

   Have a look at the resulting files, looking for the values you are interested in: time, input, output.

Repeat the data collection process in the following noise-augmented cases:

$\mathcal{B}_{pn}$ The drone is affected by disturbances that slightly change its path. To simulate this kind of disturbance, some random noise can be added on the actuator. Find `Drone` in V-REP's scene hierarchy and double-click on the slider icon to the right of the name. You can set the value for actuatorNoise (default to 0.0) and see how it affects the drone stability (0.1 is a good starting place).

Just run the simulation with a few values to choose a suitable noise amplitude. Plot the system's output `/vrep/drone/out_vel/twist/linear/x` in real time to verify that it is not drowned in the noise.

This simulates uncontrollable inputs, also known as process noise.

$\mathcal{B}_{mn}$ The readings of the output are affected by sensor noise. To simulate this noise, you can play on the measurementNoise parameter. Plot the output to select a sensible value.

This simulates measurement noise on the output.

2 Choose a model structure and identify model in the structure

In order to propose a model for our drone $\mathcal{B}$, $\mathcal{B}_{pn}$, $\mathcal{B}_{mn}$, we need to have a sequence of synchronized inputs and outputs:

$$u(0), u(1), u(2), \ldots, u(N)$$
$$y(0), y(1), y(2), \ldots, y(N)$$

These two sequences have the same number of elements and are sampled at identical rates and corresponding times (times closer from each other than half the sampling interval). As usual in discrete time modeling, these times are denoted by integers rather than seconds.

The following preprocessing steps will assist you in extracting two such sequences from your recorded data:

1. Use your favorite text editor to remove the field `field.header.frame_id` from the logfiles containing the output signal. Indeed, Matlab's function `csvread` requires the file to contain only numeric values.
2. Use `csvread('filename',1)` to import the files' content into Matlab's workspace as a matrix, skipping the first line (header text) for the same reason that you removed the field `field.header.frame_id`.

   Additional arguments can be given to `csvread` to extract only the columns you are interested in: time, input, output. See `help csvread`. Put these columns into vectors `t_in`, `in`, `t_out`, `out`.

   Alternatively, you can extract those fields from the matrix, if you prefer to import the whole file first.
3. Timestamps are in nanoseconds: to read them easily, bring them back in seconds and shift them so that they start at time zero.

   Take care to shift `t_in` and `t_out` by the same amount! It is not to be expected that both vectors start exactly at time zero: even if they are sampled at the same rate, their sampling instants do not necessarily coincide.
4. Plot the input `in` and the output `out` versus their respective time vectors. You should recognize a plot similar to those printed in real time with `rqt_plot`.
5. Use the `interp1` function to resample the output velocity to the same time stamp than the input command.
6. If `in` and `out` are not exactly the same length, crop the longest to the dimension of the shortest.

   Indeed, if the input and the output have been recorded at the same sample rate, they might differ by up to one element, depending on when between samples the recording has been stoped.
7. Use Matlab's `find` function to locate the outliers in the output signal. Remove them by replacing them by the mean of their nearest neighbors.

The following steps are concerned with choosing model structures with which to identify the system, and calculating the best model in each of these model structures:

1. Choose a few FIR and ARX model sets, that is to say, choose orders for their polynomials. We have seen that for these model sets, the one-step-ahead prediction of the output is a linear regression of the model parameters.

   Explicit the regression vector and the parameter vector for each of the FIR and ARX model sets you have chosen.
2. Assume that the prediction errors are Gaussian with zero mean values and time-independent variances. We have seen that in this case, maximizing the likelihood was like minimizing a quadratic criterion of the prediction error.

   Prove that in the case of FIR and ARX model sets, this quadratic criterion is the least-squares criterion. This demonstrates that in this special case, the maximum likelihood estimate boils down to the least squares estimate.
3. Implement a least-square optimization, using the data you have collected, to calculate the best parameter vector for each of your chosen models.

If you have validation data, check how well your identified models perform, by comparing the recorded output in the validation dataset with the output simulated by your identified models from the recorded input in the validation dataset.

You can also fire up Matlab's System Identification toolbox and run its estimation algorithms to compare the parameter values it proposes with the values you found. Choose "Polynomial Models" in the "Models" drop-down menu to select FIR, ARX and ARMAX model sets of various orders.

3 Use the model

In this last part, we will use one of your identified models to realize disturbance detection on the drone. The situation is as follows: we can use the knowledge of the drone's model, that you just identified, to compare periodically the actual output and the output simulated from the input by the model.

- If they are reasonably the same, we can deduce that the model is well-adapted to the current behavior of the drone.
- If they are suddenly substantially different, it means that the model has become ill-adapted to the system: there might have been a disturbance, or the system might have changed.

We will induce important changes in the drone by slightly changing its mass. Double-click on the drone in V-REP's scene hierarchy to change this parameter (click on the "Show dynamic properties" button). The low-level height control is proportional only, so you'll see that when becoming really heavy or really light the drone cannot maintain its height any more.

Such changes do happen in practice, for instance when a payload is added to a robot. It is therefore important to be able to detect an inadequation between the system and its (previously-satisfying) model. Therefore a new, updated model can be found by a new run of the identification process.

To assist you in comparing your model with the real system, a small ROS node is provided in

`/cs-share/pradalier/cs8804_students_ws/src/model_prediction`

Copy it in your workspace and source the `devel/setup.bash configuration`.

You don't need to modify the node, but just instantiate its parameters by modifying its launch file (`launch` directory). The parameters have the following semantic:

- trigger: string, is the topic used to decide when to publish. It can be the "state" or the "command" (input).
- command_type, state_type: the ros type for the command or state topic (e.g. std_msgs/Float64, geometry_msgs/Twist).
- command_coef_csv, state_coef_csv: the coefficients for each term as a string of comma separated value.
- command_field, state_field: subfield of interest in the command or state topic e.g. "angular.z" for a twist, "data" for a Float32, ...

Once the launch file is started, use rxplot to compare the simulated output with the reality. Try changing the system characteristics (e.g. its mass) and see if you can identify this change in the quality of the prediction.

4 Deliverable

Submit a report on the drone system identification (no ROS package here). This should include system identification results and validation, as well as graphical evaluation of the prediction performance.

This should be a short report (3-5 pages) focusing on graphical evaluation and insight gained on the process of system identification. Here are examples of questions you should ask yourselves: what works? what does not work? what are the limit of the identification process? what is difficult? what is sensitive to the parameters? is it repeatible? do we get the same results with the same data-set?