Projects for Control of Underactuated Systems (summer 2017)
Tomasz Gawron

General remarks

- The skills listed as welcomed are not mandatory to begin the project, but must be acquired to some extent during its course for a succesful result.
- Every project should be realized by a pair of students.
- The students should always report their results after finishing a milestone. Students must always consult the supervisor before moving to the next milestone.
- At the end of the project, the students should deliver a written report (paper version and a PDF file, no more than 10 pages) along with a link to a private Bitbucket git repository containing all the source code and documentation created during the project. Systematic maintenance of the repository right from the beginning of the project is highly encouraged.
- The students should report simulation/experiment results and their constructive observations attempting to explain their findings. Reports created by copying of theoretical information available in the lecture slides and textbooks will be rejected.
- Exceptionally poor language and editing of the reports will decrease the final grade.
- Milestones have grade annotations. To pass the class, one must finish and properly document all the milestones up to the one marked by grade 3.0.
- The deadline for finishing the projects and delivering the documentation is the day of the last class. *No extensions are planned.*
- Class attendance is mandatory. Students arriving later than 15 minutes after the class is scheduled to begin will not be let in.
- No food and drinks are allowed in the laboratory. Furthermore, no jackets and coats are allowed.
- Students can work in the laboratory after the classes if their testbed is available.

---

1. Optimal control for jumping robots using deep Q-learning based approximate dynamic programming

   Sketch of the problem: Feedback control for stabilizing gaits of jumping robots is hard to synthesize utilizing formal methods due to strongly nonlinear dynamics of the system, impact effects and input torque constraints. After discretizing the system, an optimal controller can be obtained using dynamic programming and Bellman's optimality principle. However, jumping robots require fine discretization, leading to intractable problem space dimensions and high computational costs. To mitigate this problem, approximate dynamic programming methods were introduced. They incrementally develop an optimal value function, which is traditionally computed through dynamic programming. In this project, the value function will be approximated using a sparse, deep neural network based representation. The network will be trained utilizing a reinforcement learning paradigm, i.e., a large number feedback control scenarios will be simulated and at the end of each simulation a reward function value will be computed. The reward function will guide the process of learning the actual value function corresponding to robot control policy.

   Skills required:

   - Matlab programming
   - Basics of nonlinear dynamical systems modelling
   - Statistics and basics of systems identification
   - Patience and passion for parameter tuning and testing various system structures

Skills welcomed:

- Python programming
- Neural nets background
- Nonlinear optimization basics
- Access to a modern GPU

Recommended tools:

- Ubuntu 16.04 with Python 2.7x
- Keras-RL library (github version)
- Theano library (github version)
- PyMatlab and Matplotlib (pip version)
- Matlab 2015a or later for Linux (64 bits)

Project milestones:

- Prepare the software setup. Test the acrobot swing-up controller example from Keras-RL.
- Analyze the acrobot example along with the supplied reinforcement learning primer.

3.0 Prepare a documented model of reward function, simulation scenario, and neural net structure. Ask the supervisor for pointers and initial solutions. Assume discrete input space with 3 values.

- Test the communication with Matlab. Implement the reinforcement learning strategy using Keras-RL, PyMatlab and supplied m-script model of the robot.

3.5 Prepare a visualization of the robot using Matplotlib and supplied Matlab example.

4.0 Train the reinforcement learning model. Obtain the basic results from a few validation runs.

4.5 Retrain the model for more complex reward functions. Move on to a finely discretized input space.

5.0 Introduce a continuous input space and/or box state constraints (i.e., robot can jump no higher than a fixed value). Retrain the model and discuss the findings.

---

2. Feedback control for car-like robots using Control Lyapunov Functions and quadratic programming

Sketch of the problem: In this project a locally optimal MPC-like (MPC stands for Model Predictive Control) controller will be designed. A Control Lyapunov Function will be proposed, from which a controller will be derived by an online solution of the particular quadratic programming problem. Such an approach allows to retain formal guarantees beneficial for feedback control, while allowing for flexible shaping of transient states, optimization of auxiliary performance indices, and consideration of practical input and state constraints.

Skills required:

- Matlab programming
- Nonlinear dynamical systems modelling
- Basics of Lyapunov stability theory
- Linear programming and basic optimization problem modelling

Skills welcomed:

- Knowledge of Control Lyapunov Functions
- Experience with LQR controllers and general quadratic programming

Recommended tools:

- Matlab
- YALMIP toolbox
- MPT toolbox (optional)

Project milestones:

- Solve a toy quadratic program in YALMIP to verify the software and learn the modelling API.
- Prepare a Control Lyapunov Function (CLF) for point stabilization of a double integrator in SE2.
- Model the quadratic program computing the control signals. Introduce soft constraint for Control Lyapunov function conditions.

3.0 Introduce auxiliary optimization of control signals energy. Introduce state constraints restraining the robot to a convex polygon.

3.5 Implement a car-like kinematics model in Matlab and test its open-loop performance. Prepare an inner feedback control loop transforming car control input to unicycle control input.

4.0 Derive a CLF on the basis of VFO convergence vector field feasible for unicycle kinematics. Convert the previously developed quadratic program (*along with all the constraints*) to car-like kinematics. Perform extensive tests of the controller.

4.5 Augment the system model with the so-called half-car dynamics. Reflect the introduced dynamics in the quadratic program.

5.0 Extend the results to prepare a quadratic program generating a control signal according to a convex combination of two CLF's.

---

3. Massively parallel, level-set based estimation of positively-invariant funnels for feedback controllers

Sketch of the problem: The knowledge of funnels describing positively-invariant state-space subsets for closed-loop control systems is essential for safe maneuvering in cluttered environments. In general, formal methods provide very conservative estimations of funnel shapes and sizes. Less conservative estimates are often obtained by using numerical methods such as sampling-based algorithms. In this project, an ability to perform highly parallel tensor manipulations on modern CPU's and GPU's will be leveraged to efficiently estimate funnel boundaries using Lyapunov functions and level-set methods. Due to fine discretization steps allowed by availability of significant computational power, a very simple conservative discretization scheme will be used.

Skills required:

- Python programming (can be learned during the project if students demonstrate proficiency in other languages)
- Strong computer science background
- A passion for debugging and code analysis

Skills welcomed:

- Familiarity with wave propagation, Dijkstra-like or Fast Marching algorithms
- Experience with data-parallel computations and CUDA programming
- Access to a modern GPU

Recommended tools:

- Ubuntu 16.04 with Python 2.7x
- Tensorflow
- Graph-tool
- Matlab for Linux

Project milestones:

- Solve a toy PDE (Partial Differential Equation) example in Tensorflow.
- Derive propagation equations used to compute level sets of supplied Lyapunov function. A simple double integrator plant is assumed.

3.0 Implement the equations in Tensorflow.

- Develop (in cooperation with the supervisor) an algorithm for reconstruction of funnel from a tensor containing Lyapunov funcion level sets. Implement a procedure checking if a point in state-space belongs to a funnel.

3.5 Perform extensive tests for various parameters of the Lyapunov function.

4.0 Use the previous results and Graph-tool to create randomized graph of connected funnels in the state space.

4.5 Develop and test a procedure estimating state space coverage by reduction/pooling of tensors with level sets for multiple Lyapunov functions.

5.0 Prepare a program, which incrementally generates a funnel graph until a certain state space coverage percentage is reached. Use Graph-tool to search this graph. Visualize the set of funnels comprising the found path.

---

4. Application of sum-of-squares optimization to controller synthesis for non-polynomial systems

Sketch of the problem: An ability to automatically derive provably stable control laws for nonlinear systems seems to be very desirable in practical robotic applications. However, classic sum-of-squares formulations of this problem are restricted to polynomial systems. Systems with trigonometric terms, which are often encountered in robotics are usually dealt with by considering Taylor expansion of the system. In this project, a rarely considered method of converting general nonlinear systems to polynomial systems by dynamic extensions will be leveraged to verify applicability of the sum-of-squares methodology to nonholonomic systems.

Skills required:

- Matlab programming
- Strong mathematical background, willingness to perform complex algebra and analysis
- Basics of Lyapunov stability theory
- Optimization modelling

Skills welcomed:

- Familiarity with semidefinite programming and sum-of-squares relaxations
- Experience with convex optimization solvers

Recommended tools:

- Matlab
- YALMIP toolbox
- MOSEK solver

Project milestones:

- Solve a toy sum-of-squares (SOS) program in YALMIP to verify the software and learn the modelling API.
- Prepare a parametric Lyapunov function candidate for a supplied polynomial system.
- Formulate an SOS program computing parameters of the Lyapunov function.

3.0 Solve the program and compute a point-wise min-norm controller introduced by Sonntag. Test performance of the controller.

3.5 Convert a model of unicycle kinematics to a polynomial system using a supplied dynamic extension algorithm. Document the process along with all the derivations.

4.0 Repeat the previous procedure of SOS program formulation using a supplied Lyapunov function structure and converted unicycle kinematics.

4.5 Compute a point-wise min-norm controller and verify its performance.

5.0 Assess and document the ability to use SOS cost functional optimization to shape controller performance.

---

5. Planning VFO feedback control policies in cluttered environments using Stable Sparse RRT's

Sketch of the problem: Stable Sparse RRT's have been developed to address the problems with optimal sampling-based motion planning for nonlinear dynamical systems. Their key property is relaxation of a classic steering function (often called an extend procedure) requirement. The steering function is replaced by just a forward propagation (i.e., integration with piecewise-constant control inputs) of a dynamical system. In this project, a reference implementation of SSRRT's will be integrated with VFO feedback control policy, which makes it possible to avoid numerical integration completely and compute feedback control policy plans instead of open-loop controls. The structure of SRRT algorithm promises to avoid some of the computational bottlenecks associated with nearest neighbor computation stemming from the structure of nonholonomic systems.

Skills required:

- C++ programming
- Computer science background
- Basic motion planning knowledge

Skills welcomed:

- Experience with graph search algorithms
- Knowledge of RRT-like planning algorithms

Recommended tools:

- Ubuntu 16.04
- QtCreator
- CMake

Project milestones:

- Compile supplied SRRT code. Test the planners with simple car-like dynamics.
- Implement a VFO propagation function using supplied materials and supervisor help. Assume path length minimzation.
- Implement visualization functions used by SRRT code.

3.0 Test the planner with VFO propagation using supplied occupancy grids.

3.5 Benchmark computational performance of the planner. Compare it with planner performance for open-loop control of a car-like robot.

4.0 Replace occupancy grid with a polygonal world resulting from occupancy grid triangluation. Exemplary triangulation code will be supplied. Test the correctness of the new approach.

4.5 Repeat the benchmarks to assess computational performance gains.

5.0 Modify assumed cost functional to consider distance to obstacles.

6. Optimizing transient behaviors of VFO controller for the waypoint-following task

Sketch of the problem: Recent extensions of the VFO controller for the waypoint-following task provide guaranteed satisfaction of various constraints and smooth transitions between execution of subsequent waypoints. However, the character of those smooth transitions is dependent on various design parameters and particulars of a given motion scenario. In this project, reinforcement learning techniques will be utilized to propose a self tuning algorithm for those parameters, which shall result in optimization of VFO transitions according to an assumed cost functional.

Skills required:

- Matlab programming
- Statistics and basics of systems identification
- Patience and passion for parameter tuning and testing various system structures

Skills welcomed:

- Python programming
- Neural nets background
- Access to a modern GPU

Recommended tools:

- Ubuntu 16.04 with Python 2.7x
- Keras-RL library (github version)
- Theano library (github version)
- PyMatlab and Matplotlib (pip version)
- Matlab 2015a or later for Linux (64 bits)

Project milestones:

- Test and analyze the supplied VFO implementation. Identify the gains to be tuned

3.0 Prepare a documented model of reward function, simulation scenario, and neural net structure. Ask the supervisor for pointers and initial solutions. Assume discrete input space with 3 values.

- Test the communication with Matlab. Implement the reinforcement learning strategy using Keras-RL, PyMatlab and VFO implementation.

3.5 Prepare a visualization of the robot using Matplotlib and supplied Matlab example.

4.0 Train the reinforcement learning model. Obtain the basic results from a few validation runs.

4.5 Propose at least 3 alternative reward functions considering different optimization objectives and constraints.

5.0 Convert one of the obtained models to C++ code using Theano (or prove the infeasibility of such a task).