Reactive Model-based Programming of Micro-UAVs

Eric Timmons and **Cheng Fang** and **Enrique Fernandez** and **Erez Karpas** and **Steven J. Levine**

Pedro Santana and **Andrew Wang** and **David Wang** and **Peng Yu** and **Brian C. Williams**
Model-based Embedded and Robotic Systems Group
Computer Science and Artificial Intelligence Lab
Massachusetts Institute of Technology

Abstract

In this demo, we will show how the Enterprise architecture, developed at the Model-based Embedded and Robotic Systems group over the past decades, can be used to control a micro aerial vehicle, namely a Parrot ARDrone. The ARDrone is programmed using RMPL, the Reactive Model-based Programming Language, which allows a user to control the vehicle with different levels of autonomy.

Introduction

RMPL, the Reactive Model-based Programming Language (Willams and Gupta 1999; Ingham, Ragno, and Williams 2001; Kim, Williams, and Abramson 2001) was developed in order to enable users to program autonomous spacecraft in a familiar way — using Java-like object oriented programming. RMPL combines plant model specification and a control program within a single program. The Enterprise architecture consists of a set of algorithms which can reason over RMPL programs.

During January 2015, the Model-based Embedded and Robotic Systems group taught an intensive, three week course to introduce (mostly) first-year students to various planning and autonomous systems concepts using various capabilities of the Enterprise architecture to control a micro UAV — the Parrot ARDrone. The students first learned to control the ARDrone by writing a command sequence, consisting of actions allowing them to move the UAV by X meters in some direction. Then, they were taught about localization and path planning, given access to a path planner, and allowed to instead tell the UAV to go from location A to location B. Finally, they were given access to a task planner, and were allowed to specify just the goals they wanted the UAV to achieve. The grand challenge in the course was to model an alien invasion scenario, and have the task planner find a solution and execute it.

In this paper, we describe how the Enterprise architecture is used to control the Parrot ARDrone. However, before we describe the architecture, we describe Temporal Plan Networks, the plan representation which is used by most of its components.

Temporal Plan Networks

A Temporal Plan Network (TPN, for short) is a tuple $tpn = \langle Ev, SV, DV, Ep \rangle$.

- Ev is the set of *events*. Each event $e \in Ev$ is atomic, and has no other attributes except for its identity (in the implementation events have an ID and name). Each event e is also associated with a guard condition $guard(e)$.
- SV is a set of *state variables*. Each state variable $sv \in SV$ is associated with a domain $dom(sv)$.
- DV is a set of *decision variables*, which is disjoint from SV. Each decision variable $dv \in DV$ is associated with a *finite* domain $dom(dv)$, and a guard condition $guard(dv)$.
- Ep is a set of *episodes*. An episode ep is a tuple $\langle fromE, toE, dc, sc, gc \rangle$, where:
  - $fromE$ and toE are events (referred to as the from event and the to event, respectively).
  - dc is a constraint on the duration of the episode. Formally, dc is a Boolean function from $\mathbb{R}$. We assume that all duration constraints are simple temporal constraints (Dechter, Meiri, and Pearl 1991), of the form $toE - fromE \in [lb, ub]$, for $lb, ub \in timedom$.
  - sc is a state constraint. sc describes feasible state trajectories during episode ep. Formally, sc is a Boolean function from $SV \times \mathbb{R}_{\geq 0}$.

    If sc is trivially true, then this episode is called a *temporal constraint*. Otherwise, we require that the duration is non-negative, that is $dc(x)$ is false for all $x < 0$.
  - gc is a guard condition, also referred to as $guard(ep)$, as described below.

A *guard condition*, $guard$, is a Boolean expression composed using arbitrary Boolean combinators (and, or, not) of expressions of the form $dv = v$ where $dv \in DV$ is a decision variable, and $v \in dom(dv)$.

We now turn our attention to defining what a "solution" for a TPN looks like. A *candidate solution* for a TPN $tpn = \langle Ev, SV, DV, Ep \rangle$ is a pair $sol = \langle s, d \rangle$ where:

- $s : Ev \rightarrow \mathbb{R}_{\geq 0} \cup \{\bot\}$ is a partial schedule, assigning times to some of the events (where $s(e) \neq \bot$) and not scheduling other events (where $s(e) = \bot$), and
- d is a partial assignment to decision variables, assigning to each $dv \in DV$ either some value in $dom(dv)$ or $\bot$.

We will denote the set of all possible candidate solutions (valid or not) by CS.

Another useful notion is that of a *state trajectory*. Formally, a state trajectory st is a function mapping each state variable to a function assigning it a value at any given timepoint. In other words, $st(sv) : \mathbb{R}_{\geq 0} \rightarrow dom(sv)$ is a function which takes a time point t and returns the value of state variable sv at time t. We will assume that $st(sv)$ is piecewise continuous for all state variables $sv \in SV$.

Finally, a candidate solution $sol = \langle s, d \rangle$ *admits* st iff:

- The partial schedule s assigns a time to all events whose guards hold, and only those events. That is, for all events $e \in Ev$, $s(e) \neq \perp$ iff $guard(e)$ is true under sol and st.
- The partial assignment d assigns a value to the decision variables whose guards hold, and only those. That is, for all decision variables $dv \in DV$, $d(dv) \neq \perp$ iff $guard(dv)$ is true under sol and st.
- For each episode $ep = \langle fromE, toE, dc, sc, gc \rangle$ such that the guard $guard(ep)$ is true under sol and st, the following hold:
  - $guard(fromE)$ and $guard(toE)$ are also true under sol and st
  - The duration constraint $dc(s(toE) - s(fromE))$ holds.
  - The state constraint sc holds in st. A state trajectory st satisfies a state constaint iff for all state variables, and for all time points t inside episode ep, the state constraint is satisfied by st, with time being shifted relative to the starting time of the episode. Formally, this is expressed as $sc(st(sv)(t), t - \min\{s(fromE), s(toE)\})$ holds for all time points t between $\min\{s(fromE), s(toE)\}$ and $\max\{s(fromE), s(toE)\}$, and for all $sv \in SV$.

Enterprise Architecture and Micro UAVs

The Enterprise architecture is a generic one, and has been used in other applications before, e.g., in the context of collaborative human-robot manufacturing (Burke et al. 2014). We first list the different components used in this demo, and will then describe how we applied the architecture to the Parrot ARDrone.

- RMPL Compiler
  The RMPL Compiler translates a control program in RMPL into an equivalent TPN.
- RMPL Preprocessor
  The RMPL Preprocessor is responsible for encoding the known map into RMPL. Each location in the map is encoded as a possible value for a finite domain variable. A simple path planner is used to compute the minimum traversal time between every pair of locations, and a primitive method corresponding to that traversal is generated. Finally, this path planner is also invoked when one of these primitive methods is executed.
- The tBurton Planner (Wang and Williams 2015)
  tBurton is a generative planner. Given a TPN with state constraints, it will generate a fully executable TPN, which contains primitive actions that achieve the desired state constraints.
- The Kirk Planner (Kim, Williams, and Abramson 2001)
  Kirk is a high-level planner, which can take a TPN with choices, and find the optimal set of choices which is temporally consistent.
- The Pike Executive (Levine and Williams 2014)
  Pike is responsible for making choices for the robots in the plan, monitoring execution, and dispatching actions at the appropriate times.
- Activity Dispatcher
  The activity dispatcher is responsible for invoking execution of actions that are dispatched by Pike. It reads the name of the action to be started (as a string), interprets that string, and invokes the appropriate command via the correct ROS interface.

In order to apply Enterprise to the ARDrone, we used the tum_ardone ROS package (Engel, Sturm, and Cremers 2014) for both localization and executing primitive motion commands. The only other adaptations that had to be made, other than modeling a micro UAV in RMPL, were encoding the movement of the ARDrone in RMPL. We used a YAML file that describes a known map, including names regions, and built an RMPL Preprocessor to convert from the known map specification to a finite domain variable, as described above.

References

Burke, S.; Fernandez, E.; Figueredo, L.; Hofmann, A.; Hofmann, C.; Karpas, E.; Levine, S.; Santana, P.; Yu, P.; and Williams, B. 2014. Intent recognition and temporal relaxation in human robot assembly. In *Proceedings of the Twenty-fourth International Conference on Automated Planning and Scheduling (ICAPS-14)*.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artificial Intelligence* 49(1-3):61–95.

Engel, J.; Sturm, J.; and Cremers, D. 2014. Scale-aware navigation of a low-cost quadrocopter with a monocular camera. *Robotics and Autonomous Systems* 62(11):1646–1656.

Ingham, M.; Ragno, R.; and Williams, B. 2001. A reactive model-based programming language for robotic space explorers.

Kim, P.; Williams, B. C.; and Abramson, M. 2001. Executing reactive, model-based programs through graph-based temporal planning. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-01)*, 487–493.

Levine, S., and Williams, B. 2014. Concurrent plan recognition and execution for human-robot teams. In *Proceedings of the Twenty-fourth International Conference on Automated Planning and Scheduling (ICAPS-14)*.

Wang, D., and Williams, B. 2015. tBurton: A divide and conquer temporal planner. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI 2015)*. AAAI Press.

Willams, B., and Gupta, V. 1999. Unifying model-based and reactive programming in a model-based executive. In *Proceedings of the 10th International Workshop on Principles of Diagnosis*.