A Tutorial on Runtime Verification and Assurance

Ankush Desai
EECS 219C

Outline

1. Background on Runtime Verification
2. Challenges in Programming Robotics System (Drona).
3. Solution 1: Combining Model Checking and Runtime Verification.
4. Solution 2: Programming Language with Runtime Assurance.

Background

Formal Verification (e.g., Model checking):

- Formal, sound, provides guarantees.
- Doesn't scale well - state explosion problem.
- Checks a model, not an implementation.
- Most people avoid it – too much effort.

Testing (ad-hoc checking):

- Most widely used technique in the industry.
- Scales well, usually inexpensive.
- Test an implementation directly.
- Informal, doesn't provide guarantees.

Runtime Verification

Attempt to bridge the gap between formal methods and ad-hoc testing.

- A program is monitored while it is running and checked against properties of interest.
- Properties are specified in a formal notation (LTL, RegEx, etc.).
- Dealing only with finite traces.

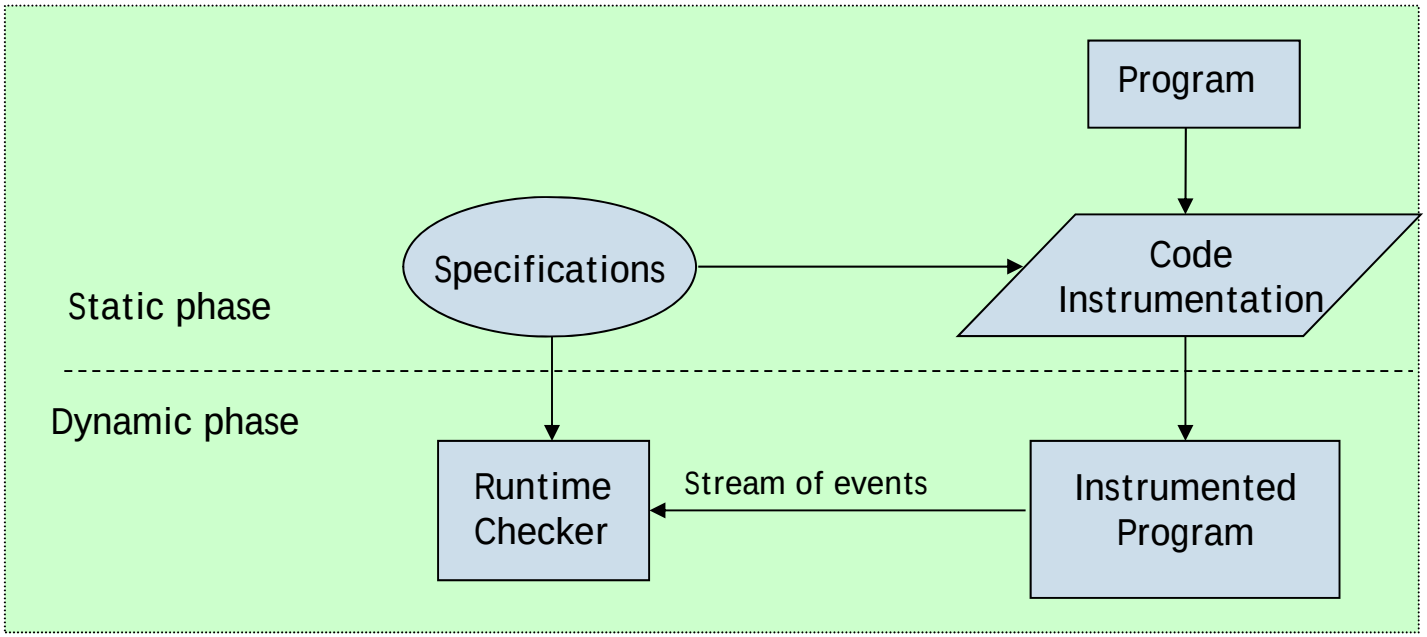
Considered as a light-weight formal method technique.

- Testing with formal “flavour”.
- Still doesn’t provide full guarantees.

Runtime Verification, cont'd

How to monitor a program?

- Need to extract events from the program while it is running.
- code instrumentation.



Still, What is Runtime Verification?

There are three interpretations of what runtime verification is, in contrast with formal verification discussed in this course.

1. RV as lightweight verification, non-exhaustive simulation (testing) plus formal specifications
2. RV as getting closer to implementation, away from abstract models.
3. RV as checking systems after deployment while they are up and running.

RV as Lightweight Formal Methods

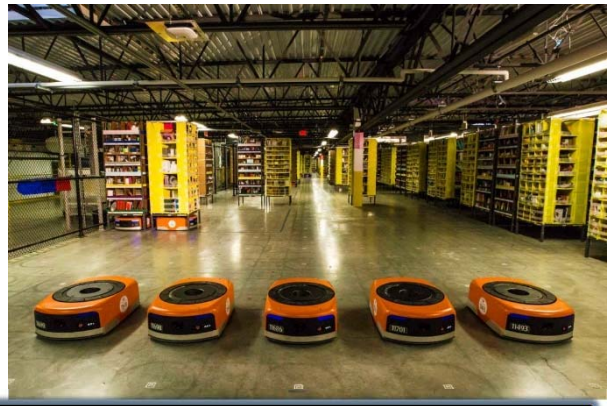
- Verification is glorious and romantic but practically hard beyond certain complexity.
- Simulation/testing is here to stay with or without attempts to guarantee some coverage.
- So let us add to this practice some formal properties and property monitors that check the simulation traces.
- Instead of language inclusion $L_s \subseteq L_\varphi$ as in verification, we check membership $w \in L_\varphi$, one trace at a time.
- Monitoring is less sensitive to system complexity. It does not require a mathematical model of the system, a program or a black box is sufficient.
- In fact, it does not care who generates the simulation traces, it could be measurements of a real physical process.

Main Challenge: Efficient monitoring

1. Low instrumentation + communication overhead.
2. An efficient monitor should have the following properties:
   - No backtracking.
     - Memory-less: doesn't store the trace.
   - Space efficiency.
   - Runtime efficiency.
     - A monitor that runs in time exponential in the size of the trace is unacceptable.
     - A monitor that runs in time exponential in the size of the formula is usable, but should be avoided.

PROGRAMMING SAFE ROBOTICS SYSTEMS

Autonomous Mobile Robotics



A major challenge in autonomous mobile robotics is programming robots with *formal guarantees* and *high assurance* of correct operation.

Delivery Systems

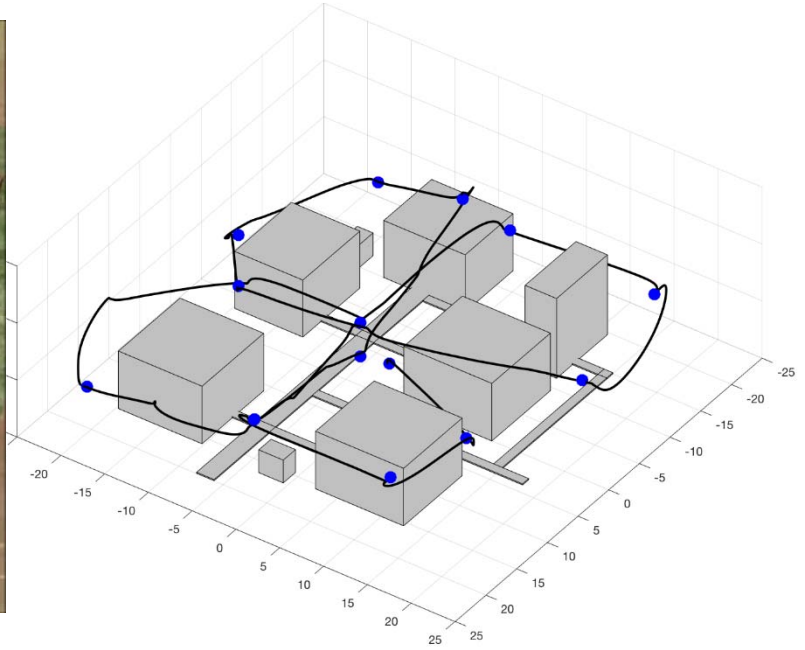
Agriculture

"*If you are able to verify a robotics system then most likely something is wrong*"

-Russ Tedrake, MIT

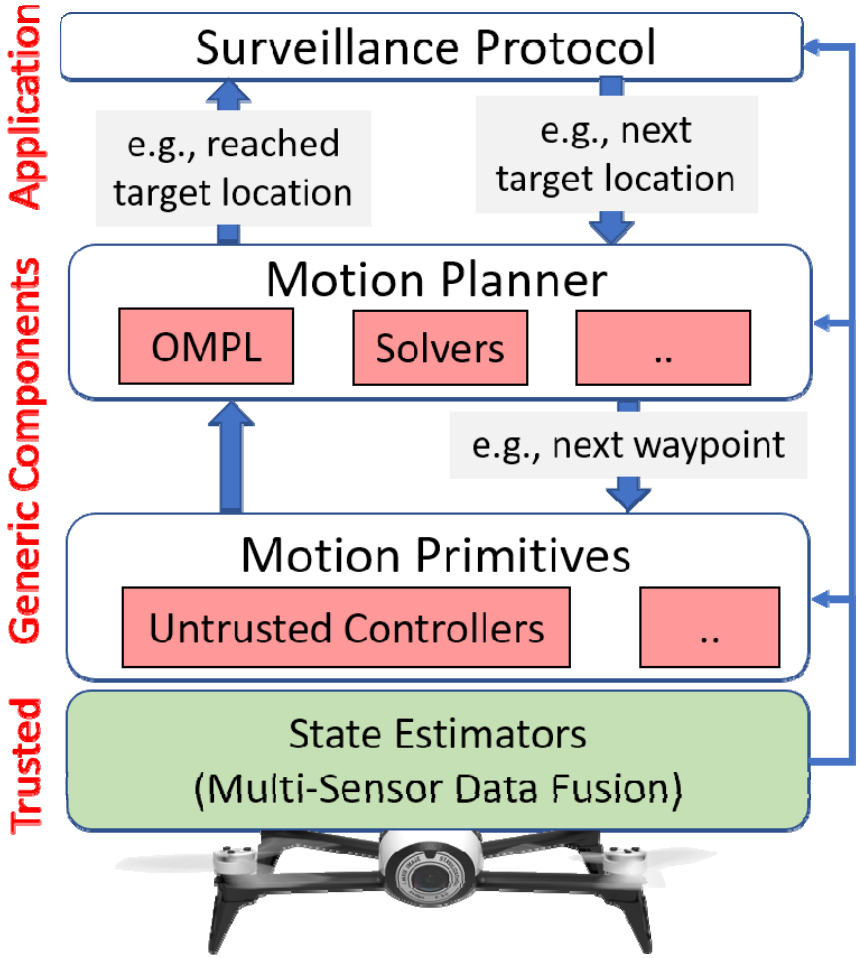
Surveillance Application

Workspace in Gazebo Simulator



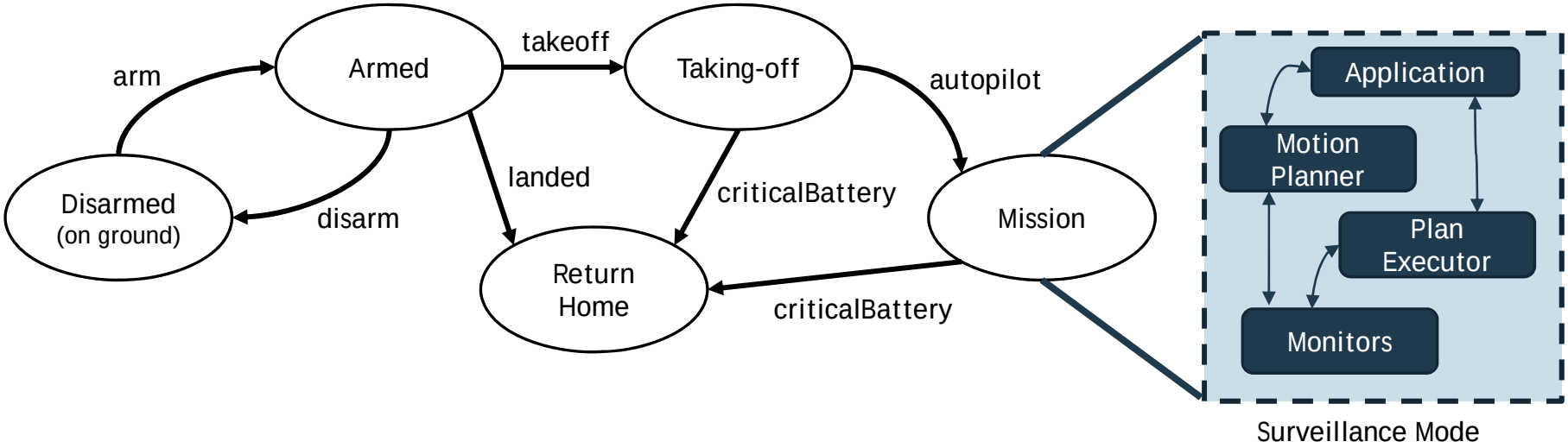
Obstacle Map and Drone Trajectory

Robotics Software Stack



Robotics Software

Robotics Software for such an autonomous drone is highly complex, reactive and concurrent.



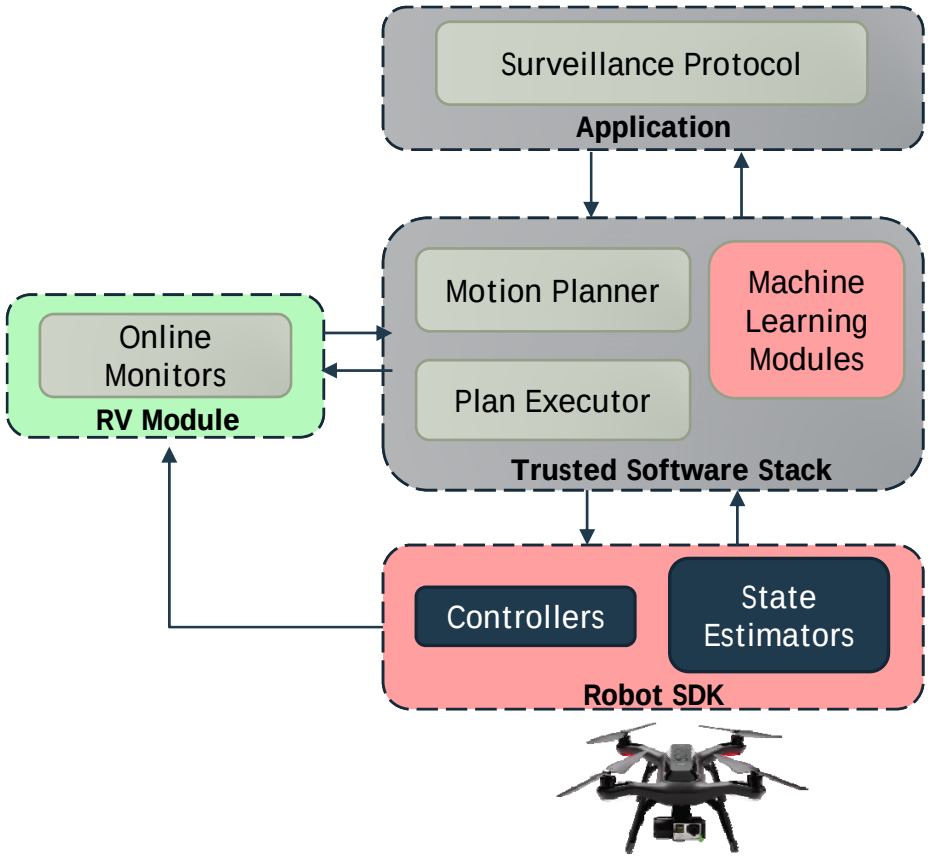
Modes of Operation of the Surveillance Drone

Our Approach

1. Programming Framework for Reactive Systems:
   - P Programming Language.

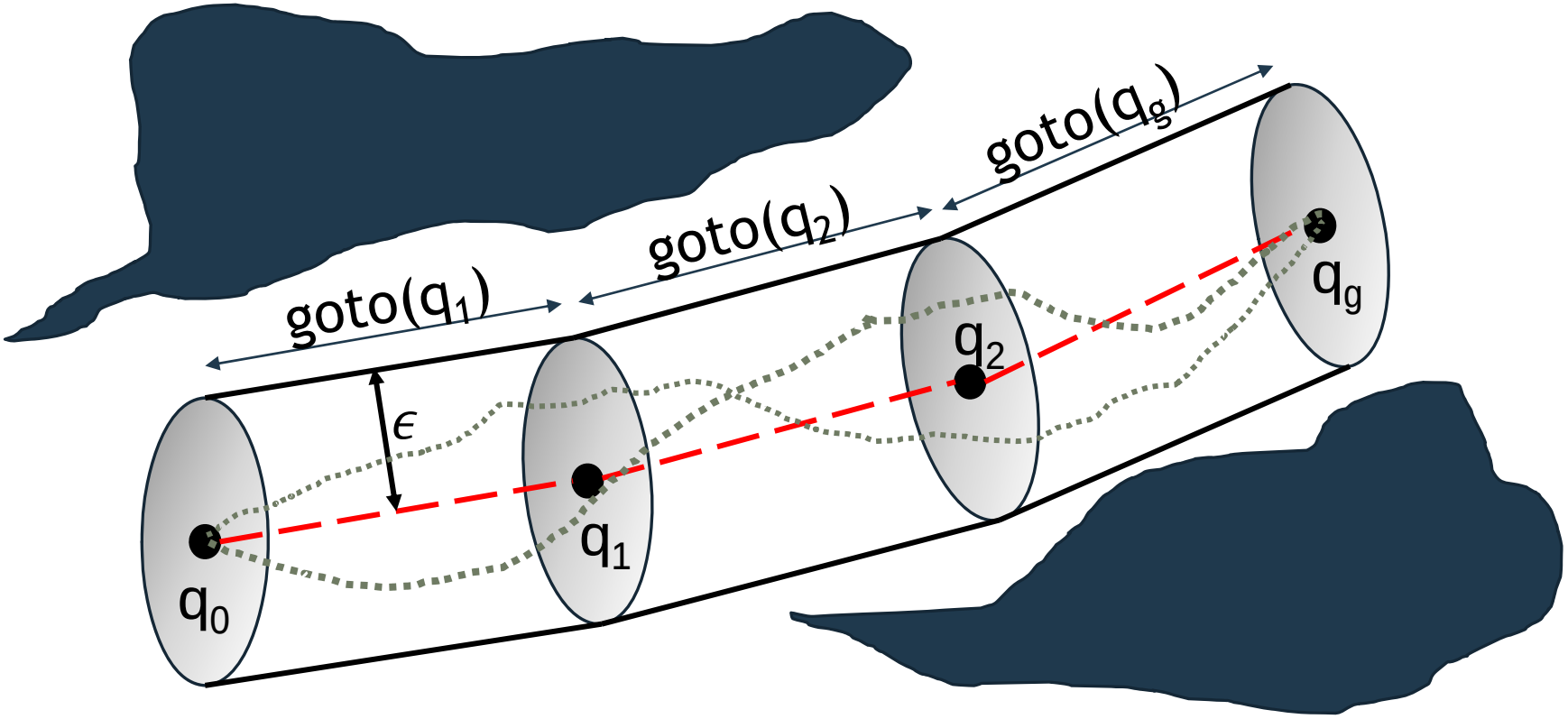
2. Scalable Analysis of Robotics Software using Model Checking.
   - Using discrete abstractions of the robot behavior.

3. Use Runtime Monitoring to ensure that the assumption hold.



Motion Planner

- Motion Primitive: goto(location)



Motion Planner

- Verify that the plans generated by the motion planner are always ϵ distance away from all obstacles.

- We used constraint solver (and RRTStar) to implement the motion planner.

Signal Temporal Logic (STL)

Syntax

- Real-time and real-valued temporal logic formulas

Semantic

- Qualitative (Boolean): Is the formula True or False?
- Quantitative (Real): How robustly is the formula True or False?

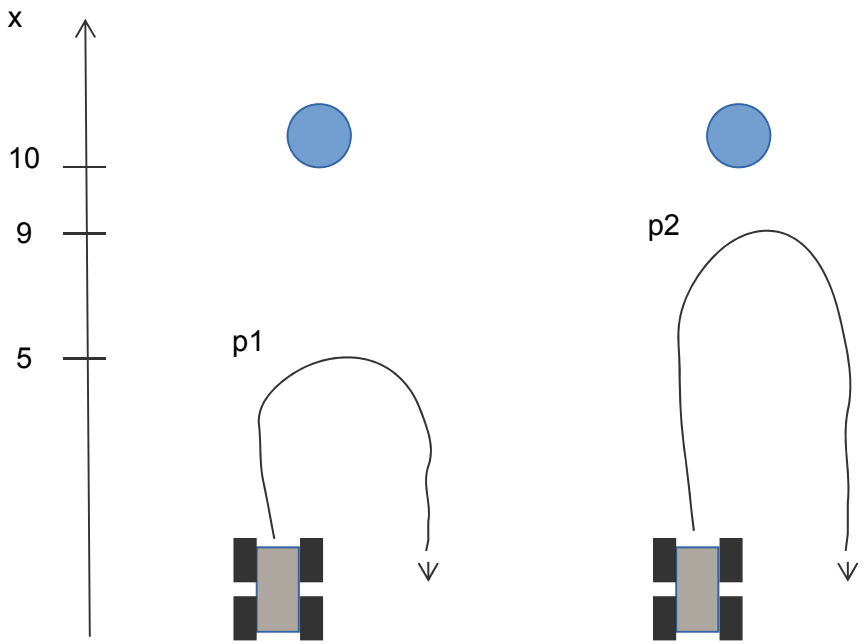
$$
\begin{aligned}
\rho^{\mu}(x,t) &= f(x_1[t],\ldots,x_n[t]) \\
\rho^{\neg\varphi}(x,t) &= -\rho^{\varphi}(x,t) \\
\rho^{\varphi_1\wedge\varphi_2}(x,t) &= \min(\rho^{\varphi_1}(x,t),\rho^{\varphi_2}(w,t)) \\
\rho^{\varphi_1\mathcal{U}_{[a,b]}\varphi_2}(x,t) &= \sup_{\tau\in t+[a,b]}(\min(\rho^{\varphi_2}(x,\tau), \inf_{s\in[t,\tau]}\rho^{\varphi_1}(x,s))
\end{aligned}
$$

Quantitative Semantics

Example

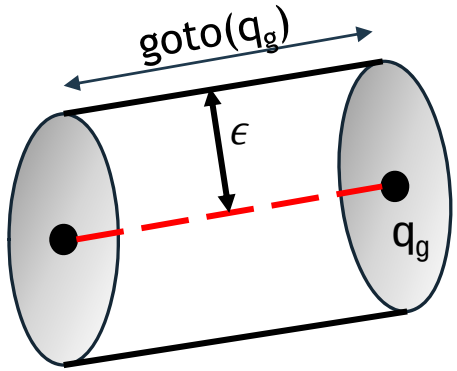
- STL requirement: Avoid an obstacle
- $f := G_{[0,5]}(x < 10)$

- Both the paths satisfy the requirement
- But p1 more robustly than p2



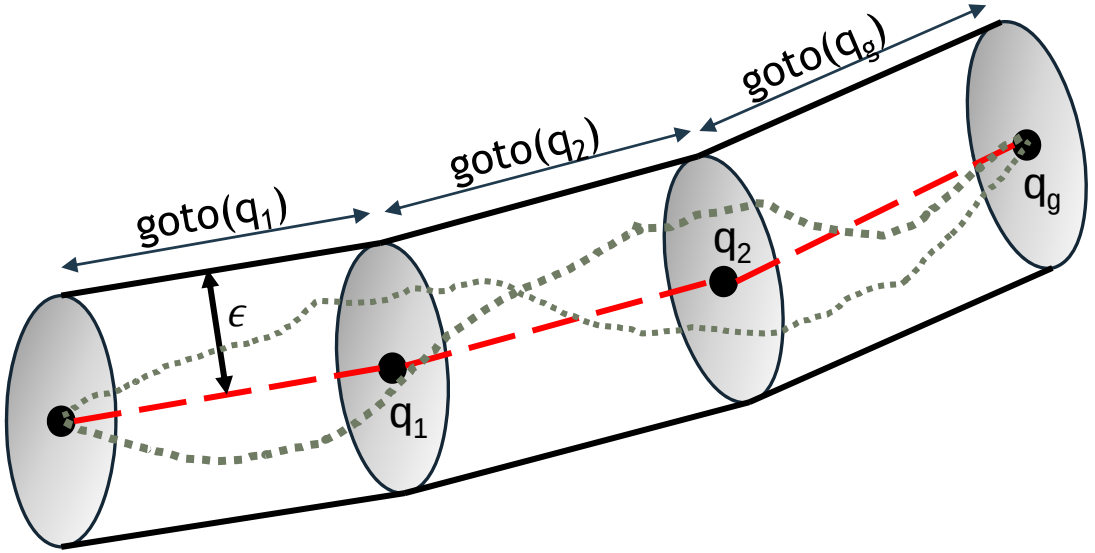
Bool:	B(f,p1) = T	B(f,p2) = T
Real:	R(f,p1) = 5	R(f,p2) = 1

Assumptions as STL Formulas



$$goto(\mathbf{q}_g, t, \epsilon) := tube((\mathbf{q}_i, \mathbf{q}_g), \epsilon) \ \mathtt{U}_{[0,t]} \ close(\mathbf{q}_g, \epsilon)$$

Assumptions as STL Formulas



$$traj(\xi, t, \epsilon) := \begin{cases} tube(\mathbf{q}_{g_1}, \mathbf{q}_{g_2}, \epsilon_1) \ \mathtt{U}_{[0,t_1]} \ close(\mathbf{q}_{g_2}, \epsilon_1) & \text{if } n = 2 \\ tube(\mathbf{q}_{g_1}, \mathbf{q}_{g_2}, \epsilon_1) \ \mathtt{U}_{[0,t_1]} \ \big(close(\mathbf{q}_{g_2}, \epsilon_1) \wedge traj(\xi', t', \epsilon')\big) & \text{otherwise} \end{cases} \tag{4}$$

where $\xi' = (\mathbf{q}_{g_2}, \ldots, \mathbf{q}_{g_n})$, $t' = (t_2, \ldots, t_{n-1})$, and $\epsilon' = (\epsilon_2, \ldots, \epsilon_{n-1})$.

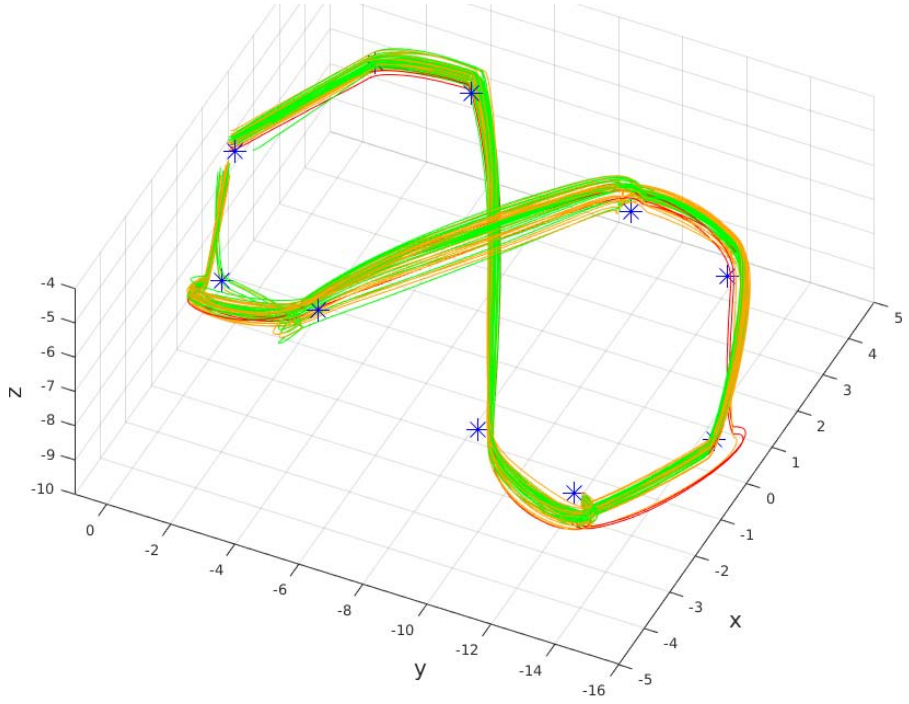
Parameter Learning

Question: What value of ϵ, t to use ?

Learn functions: f_t, f_ϵ : R x R -> R

duration $t = f_t(q_i, q_g)$ and overshoot $\epsilon = f_\epsilon(q_i, q_g)$

Validating Low-Level Controllers



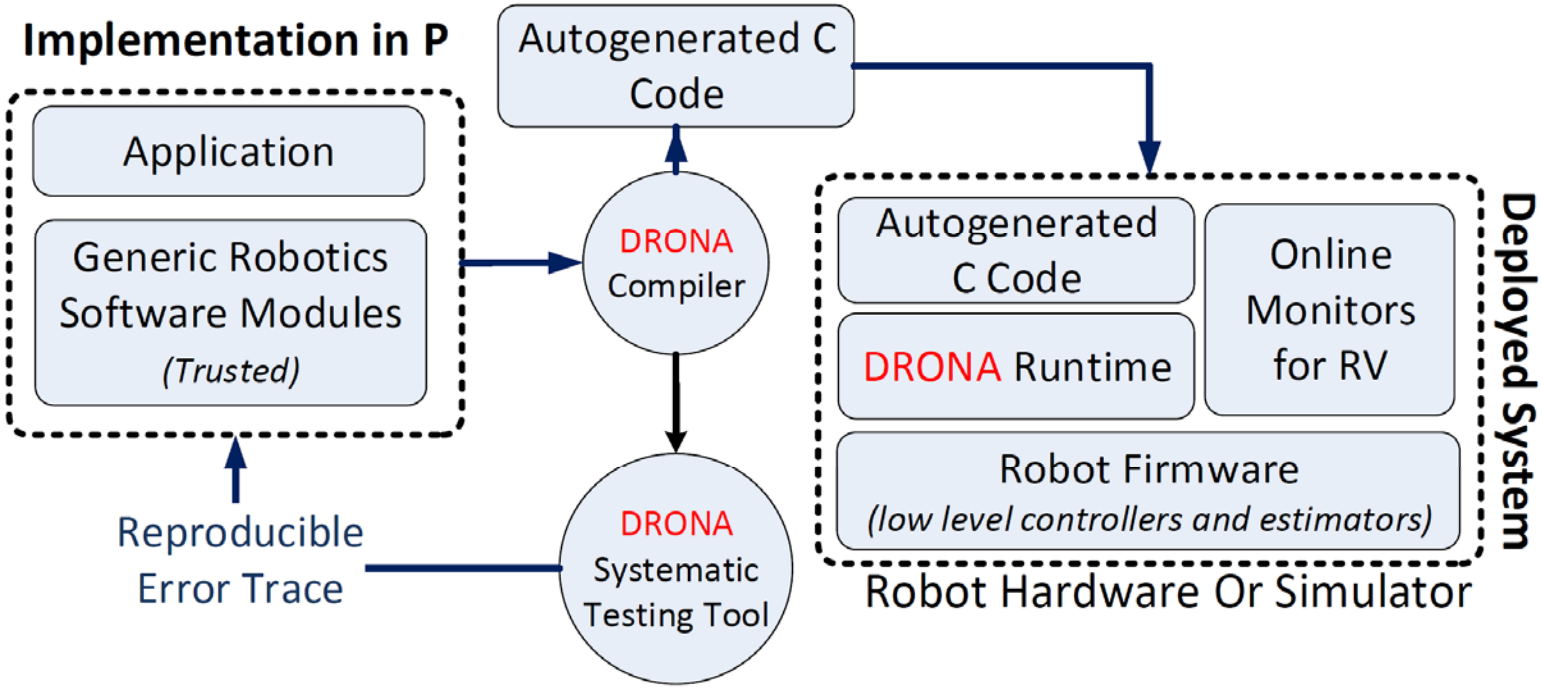
Online STL Monitoring

For each of the assumption about the low-level components:

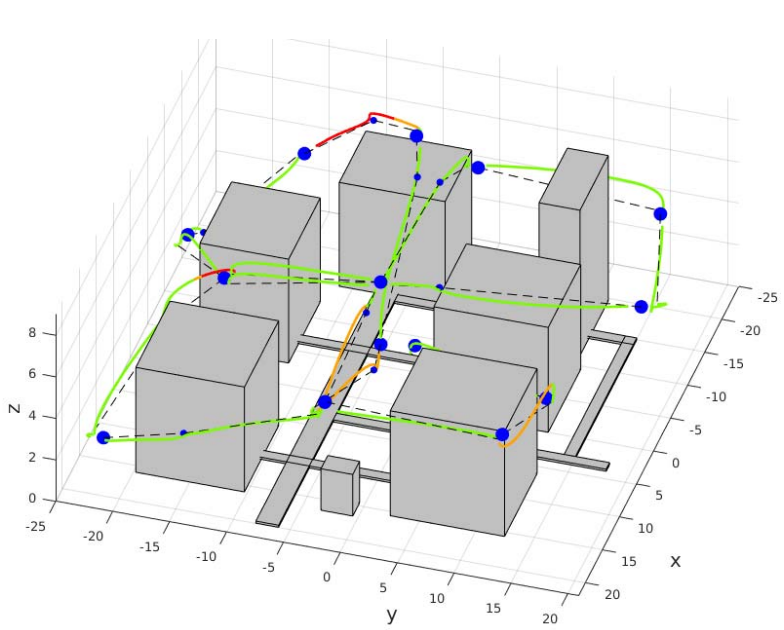
1. An STL formula is generated corresponding.

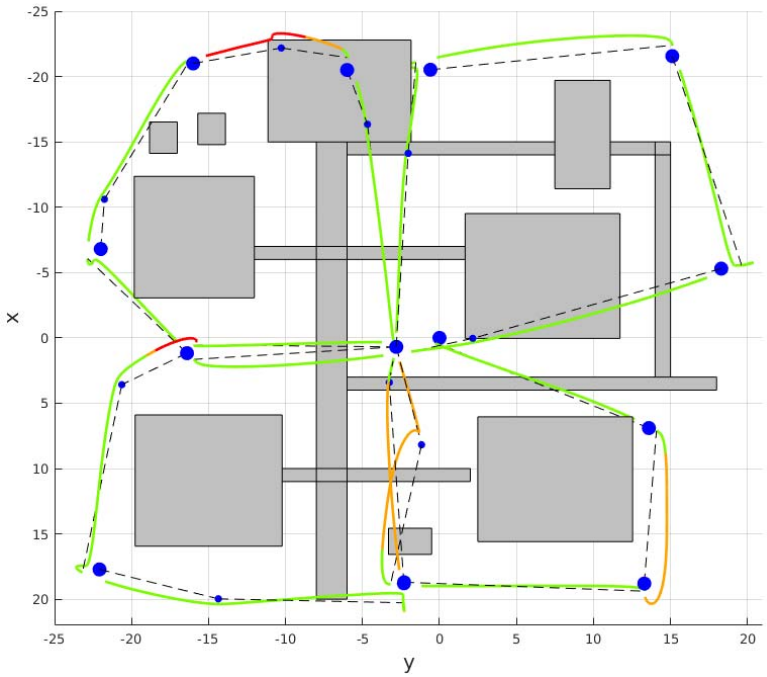
2. An online monitor is created dynamically to monitor the STL specification and take preemptive action based on the robustness value.

Drona Tool

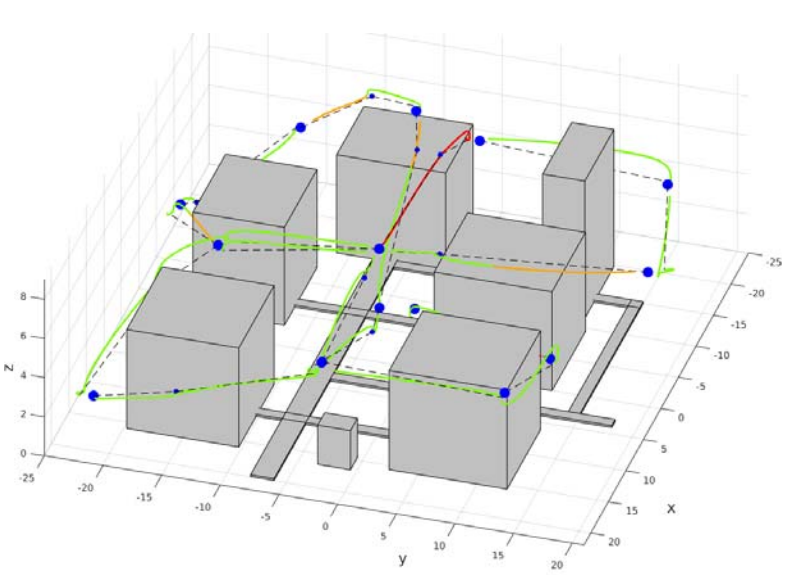

Implementation in P
Application
Generic Robotics Software Modules
(Trusted)
Autogenerated C Code
DRONA Compiler
Reproducible Error Trace
DRONA Systematic Testing Tool
Autogenerated C Code
Online Monitors for RV
DRONA Runtime
Robot Firmware
(low level controllers and estimators)
Deployed System
Robot Hardware Or Simulator

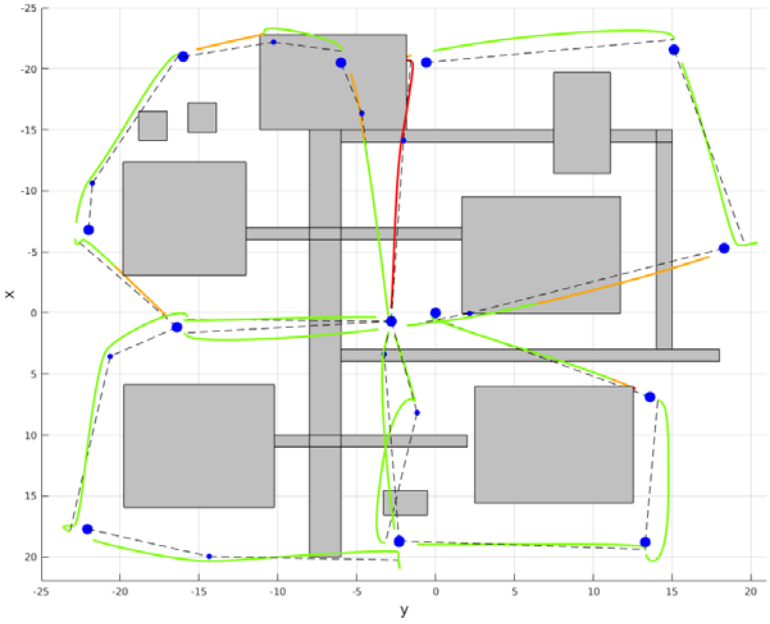
Results: Reference Trajectory





Results: Obstacle Avoidance



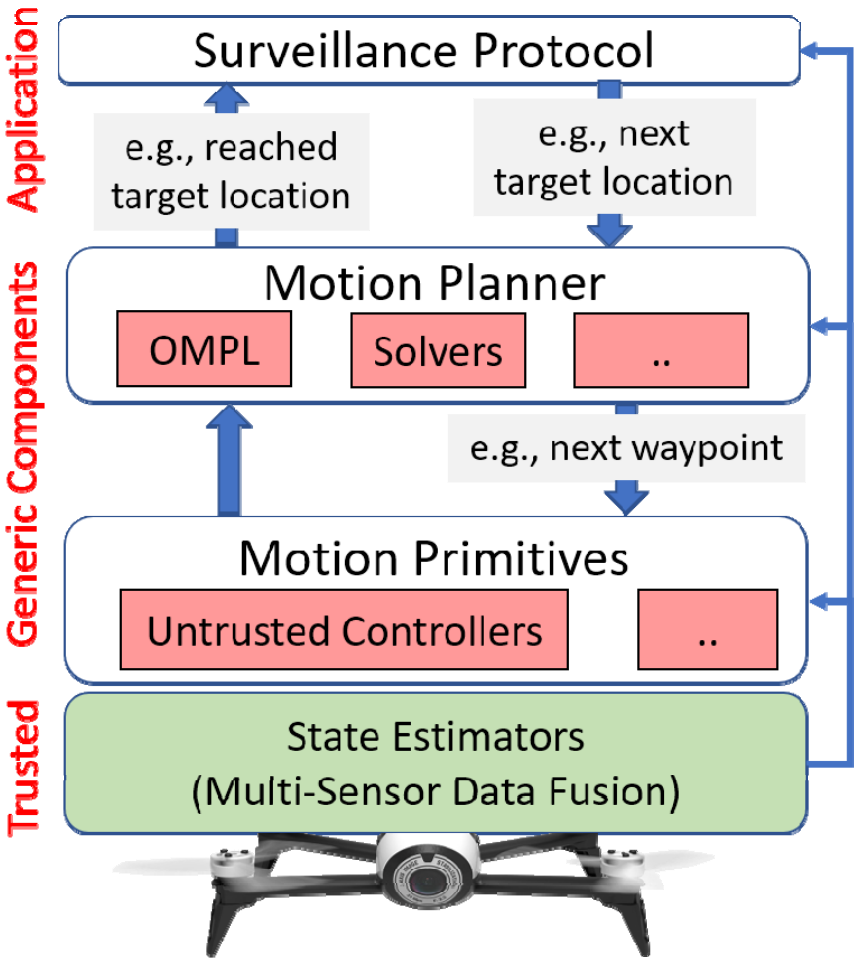


$$\varphi_{obs} := \bigwedge_{j=1}^{n} \neg \mathbf{F}_{[0,120]}(d(\mathbf{q}, obs_j) < 0.5)$$

Demo Video

RUNTIME ASSURANCE FOR SAFE ROBOTICS

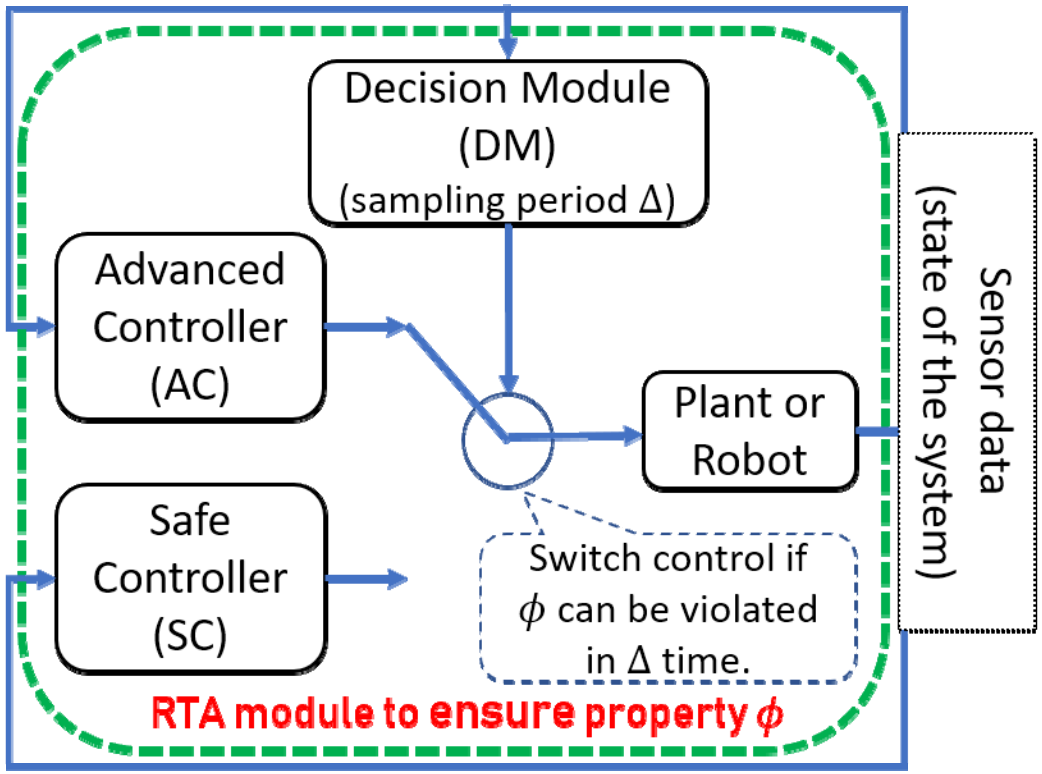
Robotics Software Stack



(1) Obstacle Avoidance (ϕ_{obs}): The drone must never collide with any obstacle.

(2) Battery Safety (ϕ_{bat}): The drone must never crash because of low battery, instead, when the battery is low it must land safely.

Simplex Architecture



A RTA Module

A RTA Module is a tuple $(Q_{ac}, Q_{sc}, \phi_{safe}, \phi_{safer}, \Delta)$

- Q_{ac} is the Advanced controller.
- Q_{sc} is the Safe controller.
- $\phi_{safer} \subset \phi_{safe}$ is a set of states.
- Δ is the sampling rate of the DM.

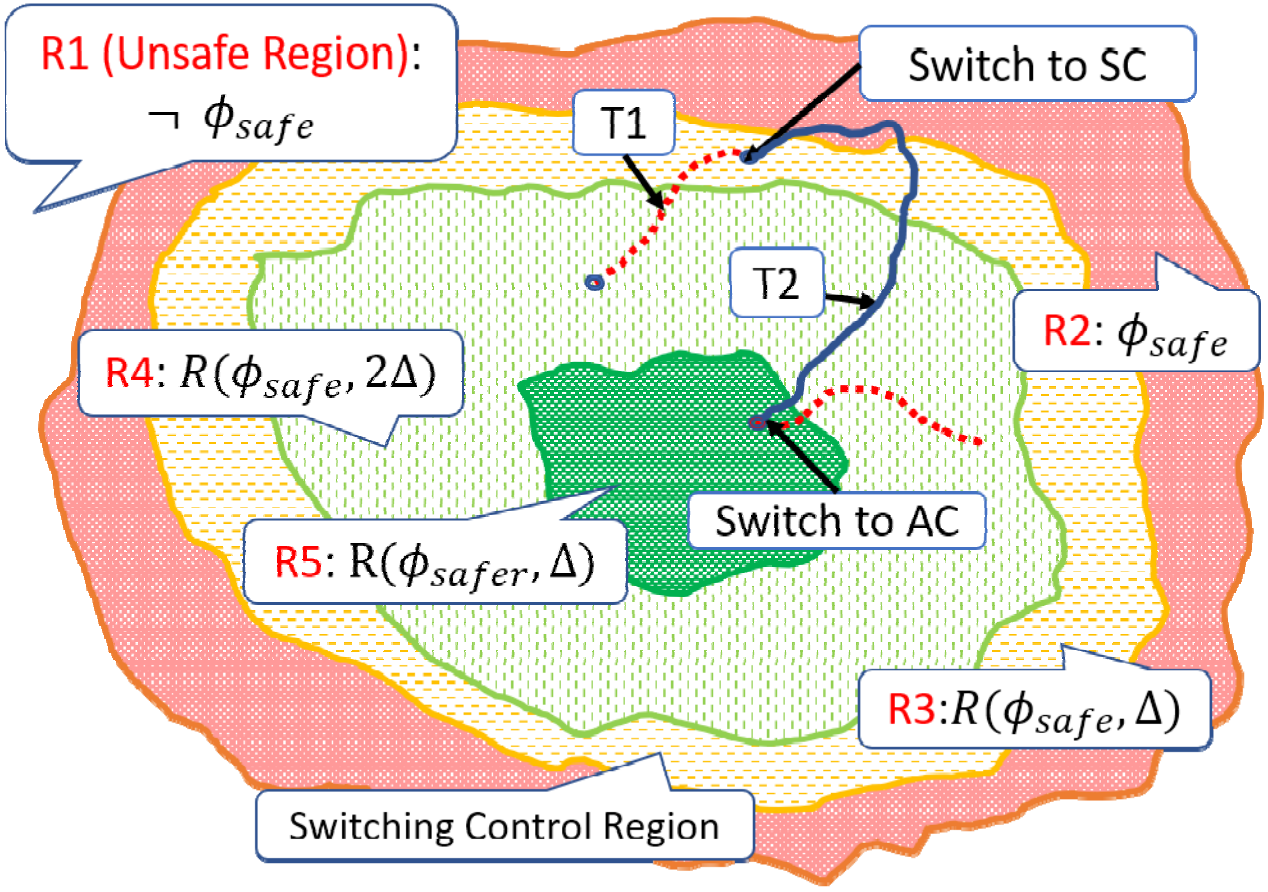
```
1 if (mode=SC ∧ s_t ∈ φ_safer) mode = AC /*switch control
      to AC*/
2 elseif (mode=AC ∧ Reach(s_t, *, 2 * Δ) ⊈ φ_safe) mode = SC
      /*switch control to SC*/
3 else mode = mode /* No mode switch */
```

A RTA machine is well-formed

A RTA Machine $(Q_{ac}, Q_{sc}, \phi_{safe}, \phi_{safer}, \Delta)$ is well-formed:

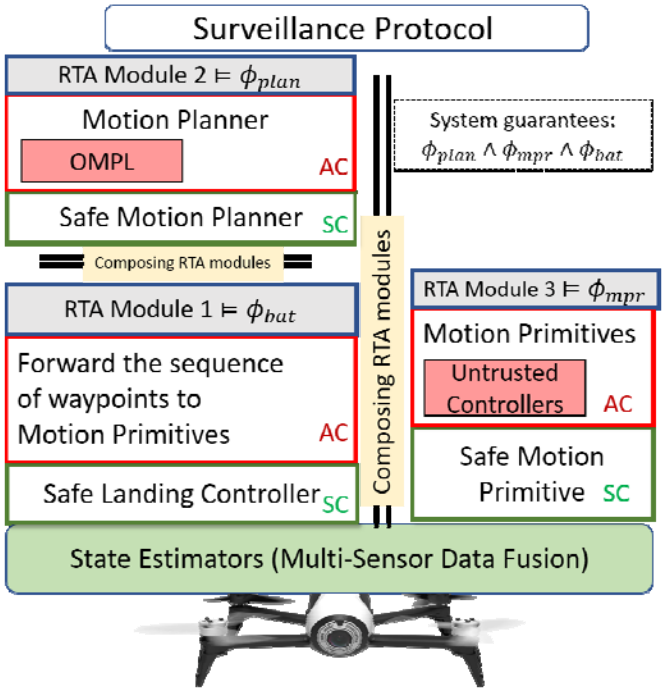
- Outputs of Q_{ac} and Q_{sc} are the same.
- Q_{ac} and Q_{sc} have same period (<= Δ).
- The Q_{sc} satisfies the following properties:
  1. $Reach(\phi_{safe}, Q_{sc}, *) \subseteq \phi_{safe}$
  2. $\forall s \in \phi_{safe}, \exists s', T \; s.t. \; s' \in Reach(s, Q_{sc}, T)$ $and \; Reach(s', Q_{sc}, \Delta) \subseteq \phi_{safer}$
  1. $Reach(\phi_{safer}, *, 2\Delta) \subseteq \phi_{safe}$. Note that this condition is stronger.

DEFINITION 4.1 (REGIONS). *Let* $R(\phi, t) = \{s \mid s \in \phi \wedge Reach_M(s, *, t) \subseteq \phi\}$. *For example,* $R(\phi_{safe}, \Delta)$ *represents the region or set of states in* ϕ_{safe} *from which all reachable states in time* Δ *are still in* ϕ_{safe}.



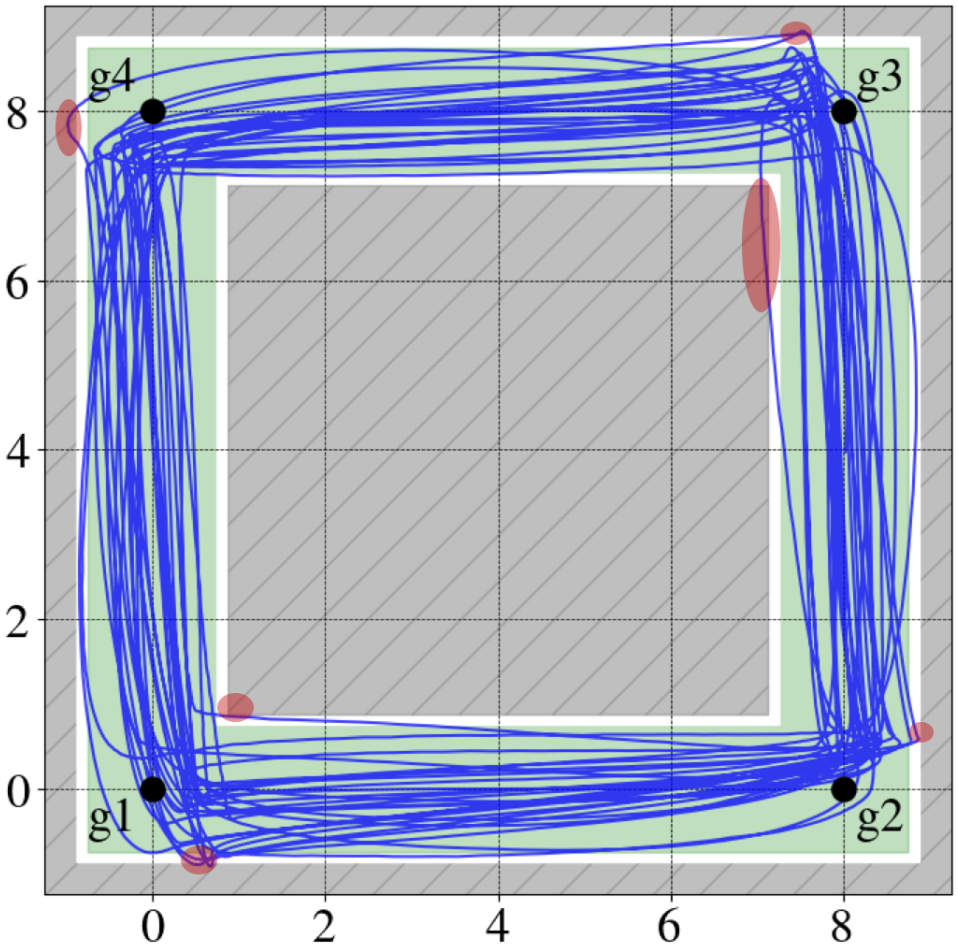
THEOREM 4.1 (**RUNTIME ASSURANCE**). *For a well-formed RTA module M, let $\phi_{\mathtt{Inv}}(\mathsf{mode}, s)$ denote the predicate* $(\mathsf{mode}{=}\mathsf{SC} \wedge s \in \phi_{safe}) \vee (\mathsf{mode}{=}\mathsf{AC} \wedge Reach_M(s, *, \Delta) \subseteq \phi_{safe})$. *If the initial state satisfies the invariant $\phi_{\mathtt{Inv}}$, then every state s_t reachable from s will also satisfy the invariant $\phi_{\mathtt{Inv}}$.*

Compositional Runtime Assurance

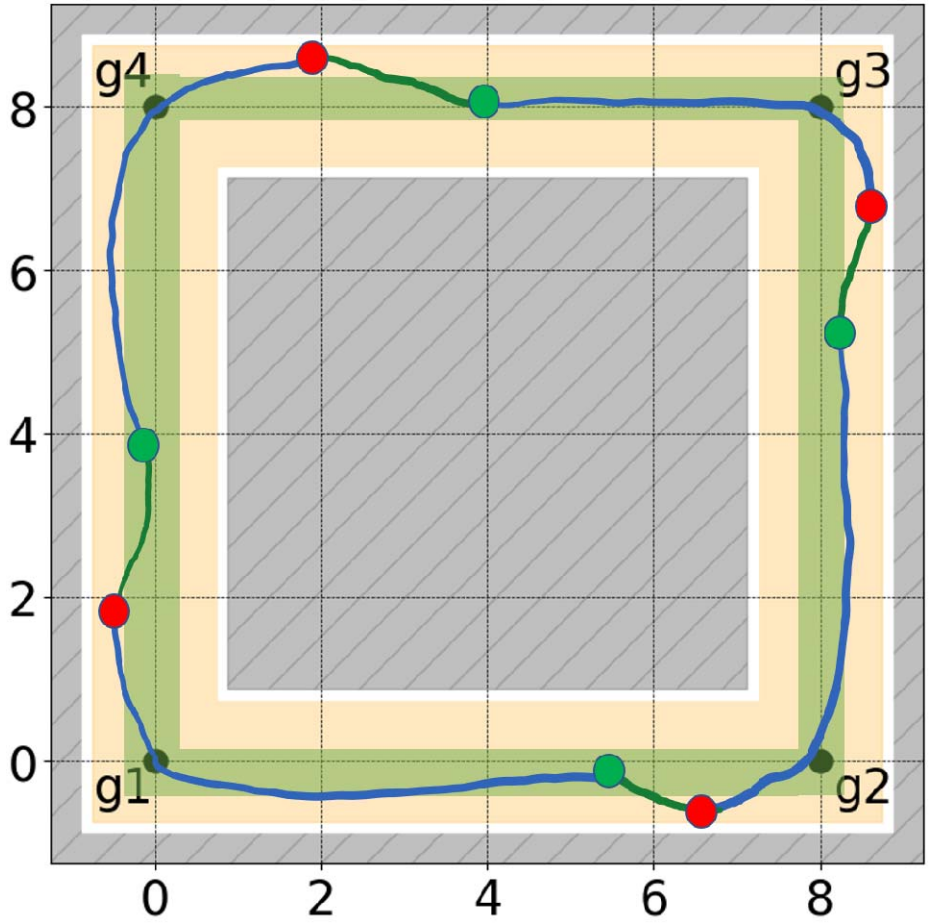


THEOREM 4.2 (COMPOSITIONAL RTA SYSTEM). *Let* $S = \{M_0, \ldots M_n\}$ *be an RTA system. If for all* i, M_i *is a well-formed RTA module satisfying the safety invariant* ϕ^i_{Inv} *(Theorem 4.1) then, the RTA system* S *satisfies the invariant* $\bigwedge_i \phi^i_{Inv}$.

Untrusted Motion Primitive



RTA Protected Motion Primitive



Only AC = 10 secs

RTA (AC + SC) = 14 secs

Only SC = 23 secs

Demo