Verification of CPS

Attack Surfaces

Most vulnerable attack surfaces in a control system are the communication channel between sensor to controller and controller to plant

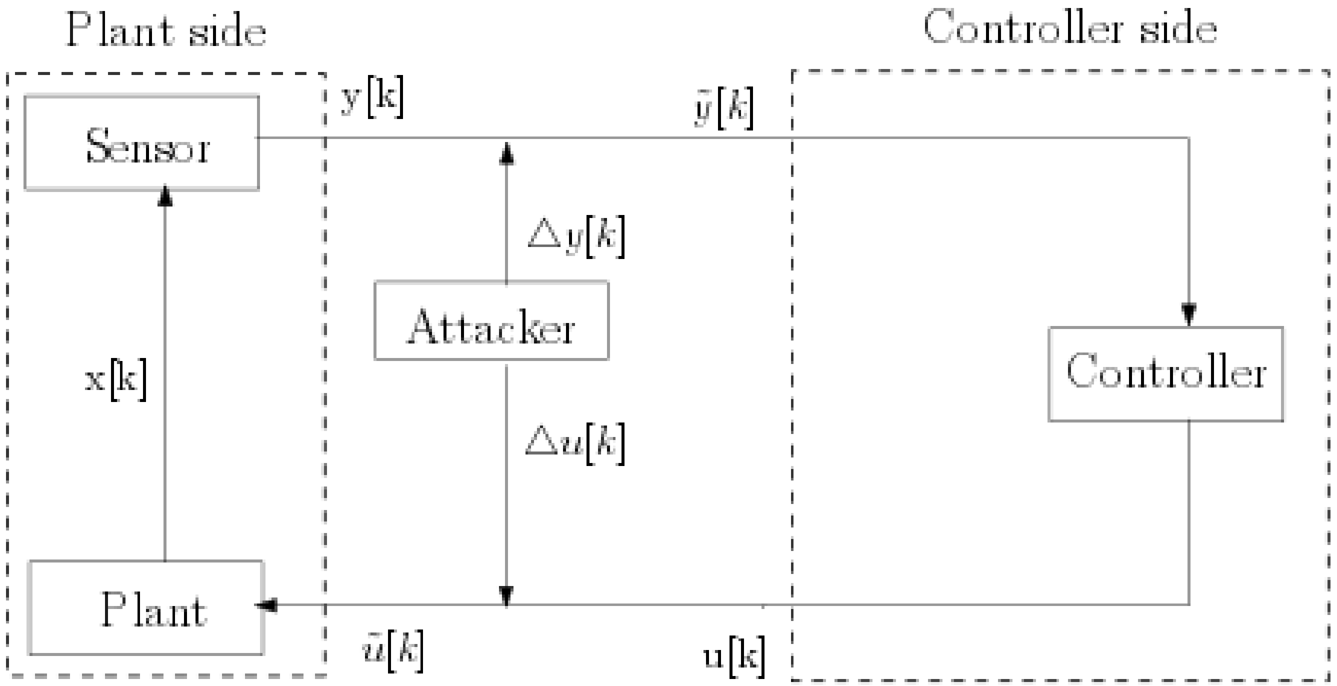


Fig: False Data Injection Attack

Need for Light Weight Detection

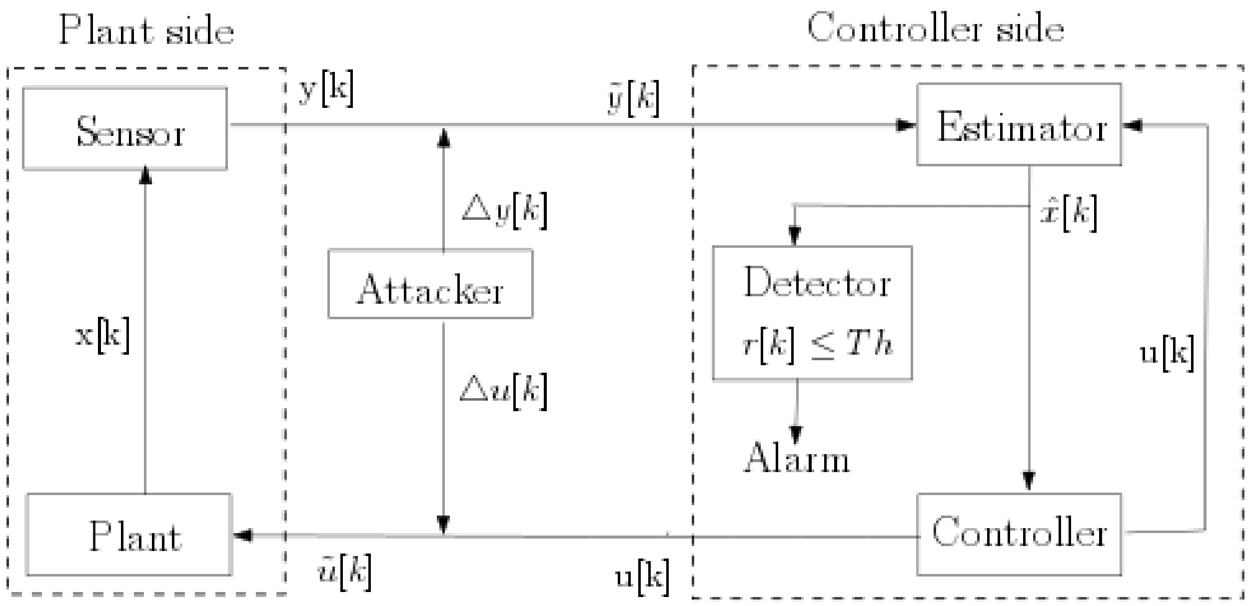
- Some safety critical CPS (like automotive) needs to satisfy real time guarantees
- Standard cryptography algorithms (like AES, RSA) can be used to secure the communication channels
- However, they incur timing overhead that may hamper the performance requirement of the system
- Hence, we need some light weight security primitives

Attack Detection: application of Kalman Filter

- Practically, all states can not be measured.
- However, to design a good controller knowledge of all system states is desired
- Kalman filter estimates the system states from the available sensor outputs such that estimation error is minimized
- The same ideology can be used to detect an attack

Attack Detection using Kalman Filter

$$r[k] = Cx[k] - C\hat{x}[k]$$
$$\hat{x}[k+1] = A\hat{x}[k] + Bu[k] + Lr[k]$$
$$x[k+1] = Ax[k] + Bu[k]$$
$$u[k+1] = -K\hat{x}[k+1]$$



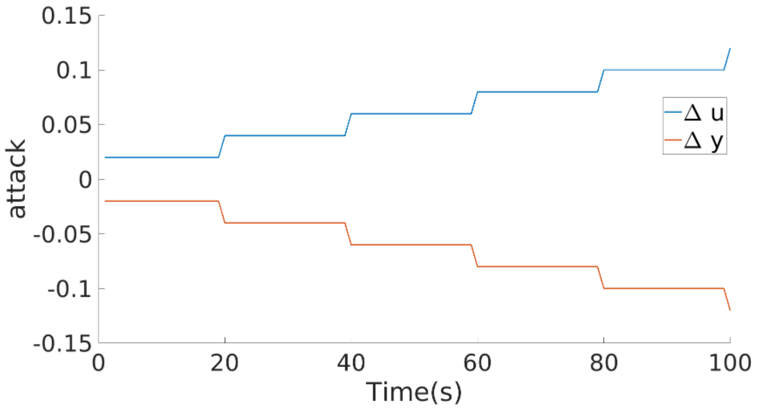
K is controller gain

L is Kalman gain

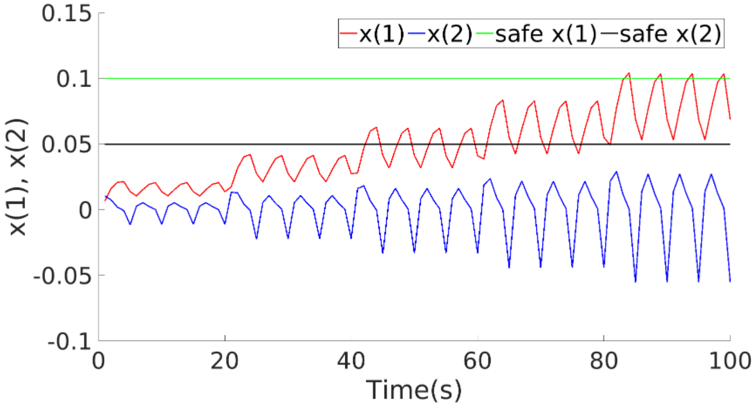
Th is predefined threshold

An attack is detected if residue r[k] exceeds the predefined threshold Th

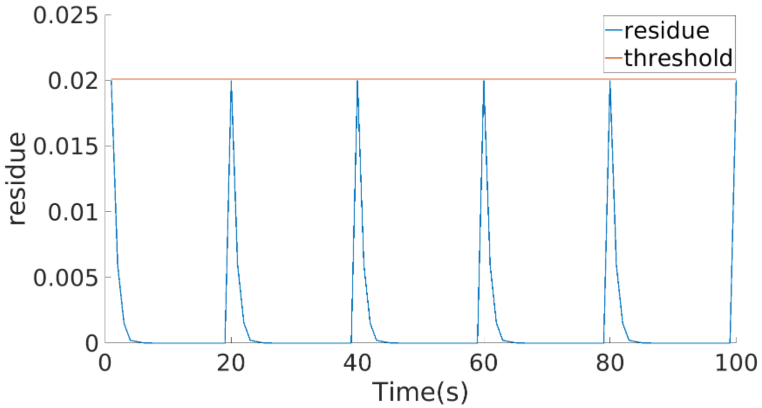
Residue Based Detector is Not Enough



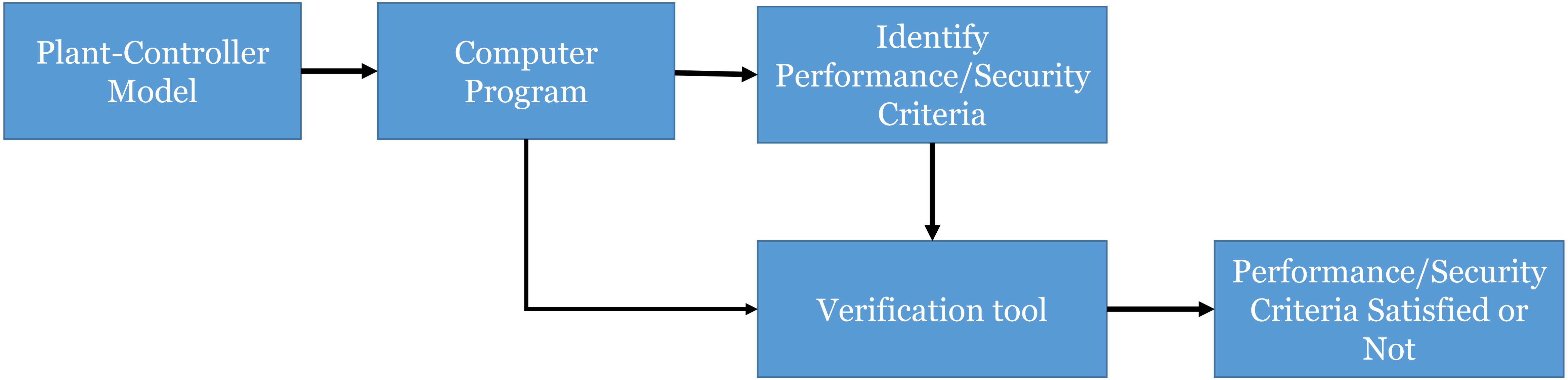
Attack which is constant over every 20 iteration drives the system to an unsafe state



However, residue always remains below predefined threshold



Verification Flow



Program Representation of Control Task

➢ $\boldsymbol{Pre-processing\ steps}$:

1. Consider a continuous plant
2. Discretize the plant $-$ controller system
3. Design discrete controller gain
4. Design observer gain (if necessary)

➢ $\boldsymbol{Initialization}$

Identify the possible range V from where the plant evaluation may start. Let $x_0 \in V$

➢ $\boldsymbol{Transforming\ the\ controller\ task}$

Let $x \in R^2$ and K = $[K_1 \quad K_2]$. Then, control input is calculated as,

$$u = -Kx = -K_1x_1 - K_2x_2$$

Program Representation of Control Task

- ***Transforming plant evaluation***

Consider $x_{k+1} = Ax_k + Bu_k$ where,

A = $\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$ and B = $\begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$

We can write plant evaluation equations as,

$$x_{k+1}[0] = a_{11}x_k[0] + a_{12}x_k[1] + b_1u$$
$$x_{k+1}[1] = a_{21}x_k[0] + a_{22}x_k[1] + b_2u$$

- ***Determining number of iterations***

Sampling period = T_s

Simulation duration = T

Number of iterations (L) = ceil(T/T_s)

Identify Performance/Security Criteria

- We define one **performance criteria** as: staring from the region V, after L iterations x should reach the region V_L

$$\boldsymbol{x_0 \in V \rightarrow x_L \in V_L}$$

$$\rightarrow \neg(\boldsymbol{x_0} \in \boldsymbol{V}) \vee \boldsymbol{x_L \in V_L} \quad \textbf{(Note A->B = } \neg\textbf{A} \vee \textbf{B)}$$

- We define one **security criteria** as: staring from region V, state of the system should always remain within safety envelop S during L iterations

$$\boldsymbol{x_0 \in V \rightarrow x_k \in S \; \forall k \in [0, L]}$$

$$\rightarrow \neg(\boldsymbol{x_0} \in \boldsymbol{V}) \vee \boldsymbol{x_k \in S \; \forall k \in [0, L]}$$

- We define the efficacy of the **detector** as: staring from region V, if residue remains below the threshold then the system must always remain safe.

$$\boldsymbol{x_0 \in V \wedge r_k < Th \; \forall k \in [0, L] \rightarrow x_k \in S \; \forall k \in [0, L]}$$

$$\rightarrow \neg(\boldsymbol{x_0} \in \boldsymbol{V \wedge r_k < Th \; \forall k}) \vee \boldsymbol{x_k \in S \; \forall k \in [0, L]}$$

$$\rightarrow \neg(\boldsymbol{x_0} \in \boldsymbol{V}) \vee \neg(\boldsymbol{r_k < Th \; \forall k}) \vee \boldsymbol{x_k \in S \; \forall k \in [0, L]}$$

Introduction to Verification Tool CBMC

- CBMC is a verification tool for ANSI-/C++ programs.
- It verifies array bounds (buffer overflows), pointer safety, exceptions and **user-specified assertions.**
- Let a C program contains an assertion clause and some non-deterministic variables. We feed the program to CBMC.
- CBMC finds assignment of the non-deterministic variables such that the assertion is violated.

Verification using CBMC

```c
float nondet_float();
int main(){
        float a,b,c;
        a = nondet_float();
        b = nondet_float();
        c = a + b;
        assert(c<=0 || c>=4);
}
```

CBMC

```
State 17 file file1.c function main line 5 thread 0
----------------------------------------------------
  a = -1.0f (10111111 10000000 00000000 00000000)

State 18 file file1.c function main line 5 thread 0
----------------------------------------------------
  b = 2.0f (01000000 00000000 00000000 00000000)

State 19 file file1.c function main line 5 thread 0
----------------------------------------------------
  c = 1.0f (00111111 10000000 00000000 00000000)
```

To generate counter example, non-deterministically chosen variables *a* and *b* are assigned with values such that assertion is violated

Verifying Performance Criteria

We represent the performance criteria $x_0 \in V \rightarrow x_L \in V_L$ in the assertion clause as:

- Let $x \in R^2$
- $x_o[0] \in [-i_o, i_o]$ and $x_o[1] \in [-i_1, i_1]$
- After L iterations $x_L[0]$ should be in $[-\Delta_o, \Delta_o]$ and $x_L[1]$ should be in $[-\Delta_1, \Delta_1]$
- Final assertion clause will be:

assert$((|x_o[0]| \leq i_o \wedge |x_o[1]| \leq i_1) \rightarrow (|x_L[0]| \leq \Delta_o \wedge |x_L[1]| \leq \Delta_1))$

$\Rightarrow$assert$(\neg(|x_o[0]| \leq i_o \wedge |x_o[1]| \leq i_1) \vee (|x_L[0]| \leq \Delta_o \wedge |x_L[1]| \leq \Delta_1))$

$\Rightarrow$assert$((|x_o[0]| \geq i_o \vee |x_o[1]| \geq i_1) \vee (|x_L[0]| \leq \Delta_o \wedge |x_L[1]| \leq \Delta_1))$

Case Study: Power Generator

States: phase angle (ϴ) and frequency deviation (ω)

Control input: normalized mechanical power

System matrices:

$$A = \begin{bmatrix} 0.66 & 0.53 \\ -0.53 & 0.13 \end{bmatrix} \; B = \begin{bmatrix} 0.34 \\ 0.53 \end{bmatrix} \; C = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Controller gain: $K = [0.0556 \;\; 0.3306]$

Observer gain: $L = \begin{bmatrix} 0.36 & 0.27 \\ -0.31 & 0.08 \end{bmatrix}$

Initial region: for ϴ [-0.1,0.1] and for ω [-0.5,0.5]

After L iteration: ϴ should be within [-0.01,0.01] and ω should be within [-0.05,0.05] after L = 15 iterations

Verifying Security Criteria

We represent the security criteria $\boldsymbol{x_0} \in \boldsymbol{V} \rightarrow \boldsymbol{x_k} \in \boldsymbol{S}\ \forall \boldsymbol{k} \in [\boldsymbol{0}, \boldsymbol{L}]$ in the assertion clause as:

- Let $x \in R^2$
- $x_0[0] \in [-i_0, i_0]$ and $x_0[1] \in [-i_1, i_1]$
- We define safety bound for x[0] and x[1] as s[0] and s[1]
- Final assertion clause will be:

assert(($|x_0[0]| \le i_0 \wedge |x_0[1]| \le i_1) \rightarrow (|x_0[0]| \le s[0] \wedge |x_0[1]| \le s[1] \wedge |x_1[0]| \le s[0] \wedge |x_1[1]| \le s[1] \wedge \ldots \wedge |x_L[0]| \le s[0] \wedge |x_L[1]| \le s[1])$)

$\Rightarrow$assert($\neg(|x_0[0]| \le i_0 \wedge |x_0[1]| \le i_1) \vee (|x_0[0]| \le s[0] \wedge |x_0[1]| \le s[1] \wedge |x_1[0]| \le s[0] \wedge |x_1[1]| \le s[1] \wedge \ldots \wedge |x_L[0]| \le s[0] \wedge |x_L[1]| \le s[1])$)

$\Rightarrow$assert(($|x_0[0]| \ge i_0 \vee |x_0[1]| \ge i_1) \vee (|x_0[0]| \le s[0] \wedge |x_0[1]| \le s[1] \wedge |x_1[0]| \le s[0] \wedge |x_1[1]| \le s[1] \wedge \ldots \wedge |x_L[0]| \le s[0] \wedge |x_L[1]| \le s[1])$)

Case Study: Power Generator

States: phase angle (ϴ) and frequency deviation (ω)

Control input: normalized mechanical power

System matrices:

$$A = \begin{bmatrix} 0.66 & 0.53 \\ -0.53 & 0.13 \end{bmatrix} \; B = \begin{bmatrix} 0.34 \\ 0.53 \end{bmatrix} \; C = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Controller gain: $K = [0.0556 \;\; 0.3306]$

Observer gain: $L = \begin{bmatrix} 0.36 & 0.27 \\ -0.31 & 0.08 \end{bmatrix}$

Initial region: for ϴ [-0.01,0.01] and for ω [-0.005,0.005]

Safety boundary: ϴ should be within [-0.1,0.1] and ω should be within [-0.05,0.05]

Introducing Detector

We represent $\boldsymbol{x_0} \in \boldsymbol{V} \wedge \boldsymbol{r_k} < \boldsymbol{Th}\ \forall \boldsymbol{k} \in [\boldsymbol{0}, \boldsymbol{L}] \rightarrow \boldsymbol{x_k} \in \boldsymbol{S}\ \forall \boldsymbol{k} \in [\boldsymbol{0}, \boldsymbol{L}]$ in the assertion clause as:

- Let $x \in R^2$
- $x_0[0] \in [-i_0, i_0]$ and $x_0[1] \in [-i_1, i_1]$
- We define safety bound for x[0] and x[1] as s[0] and s[1]
- Final assertion clause will be:

assert$((|x_0[0]| \leq i_0 \wedge |x_0[1]| \leq i_1 \wedge ||r_0|| < Th \wedge ||r_1|| < Th \wedge \ldots \wedge ||r_L|| < Th) \rightarrow (|x_0[0]| \leq s[0] \wedge |x_0[1]| \leq s[1] \wedge |x_1[0]| \leq s[0] \wedge |x_1[1]| \leq s[1] \wedge \ldots \wedge |x_L[0]| \leq s[0] \wedge |x_L[1]| \leq s[1]))$

$\Rightarrow$assert$(\neg(|x_0[0]| \leq i_0 \wedge |x_0[1]| \leq i_1 \wedge ||r_0|| < Th \wedge ||r_1|| < Th \wedge \ldots \wedge ||r_L|| < Th) \vee (|x_0[0]| \leq s[0] \wedge |x_0[1]| \leq s[1] \wedge |x_1[0]| \leq s[0] \wedge |x_1[1]| \leq s[1] \wedge \ldots \wedge |x_L[0]| \leq s[0] \wedge |x_L[1]| \leq s[1]))$

$\Rightarrow$assert$((|x_0[0]| \geq i_0 \vee |x_0[1]| \geq i_1 \vee ||r_0|| \geq Th \vee ||r_1|| \geq Th \vee \ldots \vee ||r_L|| \geq Th) \vee (|x_0[0]| \leq s[0] \wedge |x_0[1]| \leq s[1] \wedge |x_1[0]| \leq s[0] \wedge |x_1[1]| \leq s[1] \wedge \ldots \wedge |x_L[0]| \leq s[0] \wedge |x_L[1]| \leq s[1]))$

Case Study: Power Generator

States: phase angle (ϴ) and frequency deviation (ω)

Control input: normalized mechanical power

System matrices:

$$A = \begin{bmatrix} 0.66 & 0.53 \\ -0.53 & 0.13 \end{bmatrix} \; B = \begin{bmatrix} 0.34 \\ 0.53 \end{bmatrix} \; C = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Controller gain: $K = [0.0556 \;\; 0.3306]$

Observer gain: $L = \begin{bmatrix} 0.36 & 0.27 \\ -0.31 & 0.08 \end{bmatrix}$

Initial region: for ϴ [-0.01,0.01] and for ω [-0.005,0.005]

Safety boundary: ϴ should be within [-0.1,0.1] and ω should be within [-0.05,0.05]

Detector: 1-norm of the residue should be within 0.05 (i.e. the threshold is 0.05)

CBMC Guide

- **Download**: https://www.cprover.org/cbmc/
- **Manual:** https://www.cprover.org/cbmc/doc/manual.pdf
- **Necessary commands:**
  - *If C code does not contain any while loop:* ./cbmc <c file name> --trace

  Example: ./cbmc test.c --trace
  - *If C code has while loop:* ./cbmc <c file name> --trace --unwind <max unwind number>

  Example: ./cbmc test.c --trace --unwind 100

  This will unwind every while loop in test.c 100 times