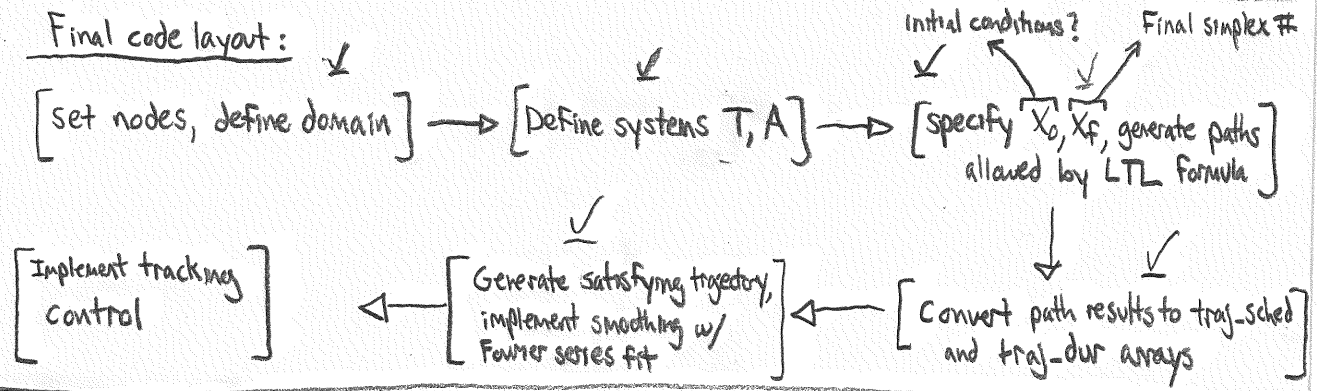
Building traj_sched and traj_dur from product automaton solution:

Final code layout:



```
def no_small_satisfaction_v1(sol_prod_states, sol_T_states, sol_trans, S1_ID):

    sol_T_out = np.zeros((sol_prod_states.shape[0], ...))
    sol_trans_out = np.zeros(...)

    for idx1 in range(sol_T_states
```

(+1)

```
sing_occ_array = np.zeros
```

```
traj_sched = np.concatenate(([sol_T_select], [control_ppt_array], [facet_idx_array]), axis=0)

r_steering = np.array([fourier_series(t, T, x_coeff[0,:], x_coeff[1,:], x_coeff.shape[1]), ...])
v_steering = np.array([fourier_series_dot(...), ...])
vdot_steering = np.array([fourier_series_ddot(...), ...])
```

(-1)

q_old = q_end

trans_idx, pred_idx = unpack_delta_back(delta_p, 39, 4, curr_idx)

inputs: arrays of state futures, trans futures → extract basket ("current" state)

For current state in basket, compute predecessor states and transitions:

For each pred. state, append to array with state future added

For each associated trans, append to array with trans future added

This function only computes history "trees". The function that calls it must process the outputs and discard branches, subject to some rule.

---

def tree_branch(future_states, future_trans, future_x): ← delta_p, Qp

For idx in range(future_states.shape[0]):

trans_labels, pred_labels = unpack_delta_fwd(delta_p, len(Qp), 4, state_now)

state_now = future_states[idx, -1]

build arrays to append to: temp_state_array = np.tile(future_states[idx, :], (, , 1))

for idx2 in range(len(pred_labels)):

len(pred_labels)

state_array_out = np.zeros((len(pred_labels), future_states.shape[1] + 1))

state_array_out[idx2, :] = np.append(temp_state_array[idx2, :], [pred_labels[idx]])

?

future_states = $\begin{bmatrix} \cdot & \cdot & * \\ \cdot & \cdot & * \end{bmatrix}$ $\quad [[V_1] \;\; V_2]$

inputs: future_states, delta_p, Qp,

IF the accepting

inputs: final_prod_idx, Qp

max_iter, recurs

To Do:
- Build functions for final computations in system_functions → extend to specify ICs??
- Build function to convert sol_T_states, sol_trans to traj_sched, traj_dur arrays
- Build a master simulation.py file
- trajectory tracking function
- figures, write-up

$$\dot{\bar{e}} = [A]\bar{e} + [B]\bar{u}$$

$$\dot{\bar{e}} = \left([A] - [B][K]\right)\bar{e} \quad \text{(closed-loop)}$$

4×4, 4×2, 2×4

$$K = \begin{bmatrix} K_{11} & K_{12} & K_{13} & K_{14} \\ K_{21} & K_{22} & K_{23} & K_{24} \end{bmatrix}$$

$$\left([A] - [B][K]\right) = \begin{bmatrix} 0_{2\times2} & I_{2\times2} \\ -K_1 & -K_2 \end{bmatrix}, \quad K = \begin{bmatrix} K_1 & K_2 \end{bmatrix}$$

(K_1: 2×2, K_2: 2×2)

$$\left|\lambda[I_{4\times4}] - [A_{cl}]\right| = \lambda^4 + (K_{13}+K_{24})\lambda^3 + (K_{11}+K_{22})\lambda^2 + (K_{11}K_{24}+K_{13}K_{22})\lambda + K_{11}K_{22} = 0$$

$$\lambda = \frac{1}{2}\left(-K_{24} \pm \sqrt{K_{24}^2 - 4K_{22}}\right), \quad \lambda = \frac{1}{2}\left(-K_{13} \pm \sqrt{K_{13}^2 - 4K_{11}}\right)$$

Need $K_{24} > 0$, $K_{13} > 0$, $K_{24}^2 - 4K_{22} \geq 0$

$$K_{13}^2 - 4K_{11} = 0$$

$$K_{22} = \frac{K_{24}^2}{4}, \quad K_{11} = \frac{K_{13}^2}{4}$$

$$\tau_1 = \frac{2}{K_{24}}, \quad \tau_2 = \frac{2}{K_{13}}$$

```
def delta_prod(Qp, delta, user_0, alph_fcns):
    delta_p = np.zeros((4 * Qp.shape[0], 3))
    for idx in range(Qp.shape[0]):
        for idx2 in range(4):
            x_idx_temp = Qp[idx, 1]                      ✓
            d_idx_temp = 4 * x_idx_temp + idx2           ✓
            x_res = delta[d_idx_temp, 2]                 ✓
            q_idx_temp = Qp[idx, 2]                      ✓
            ~~R_idx2 = 10 * q_idx_temp~~
            if x_res != -1:
                obs_res = obs_from_state(x_res, user_0)
                # From q_idx_temp, check available Boolean formulas to check w/ obs_res   ] new function ✓
                    (using R)        ↑ obtain
                # From available Boolean formulas, [J], check:        j ∈ [J]
                    alph_fcns[j](obs_res)                                                  ] new function ✓
                    -> if any are satisfied, q_res = R[10 * q_idx_temp + j, 2]   (10 = len(alph_fcns))
                    otherwise, q_res = -1
                if (x_res != -1) and (q_res != -1):
                    delta_p[4*idx + idx2, :] = np.array([Qp[idx,0], idx2, which_qp_state(x_res, q_res, Q)])
                else:
                    delta_p[4*idx + idx2, :] = np.array([Qp[idx,0], idx2, -1])
    return delta_p
```

Note we are deterministic

*** Test this in a debug section in main() before turning into a function!

```
def unpack_delta(delta_array, n_states, n_trans, state_sel):
    for idx in range(n_states):
        for idx2 in range(n_trans):
            if (delta_array[n_trans*idx1 + idx2, 1] == state_sel) and (delta_array[" ",2] != -1):
```

tri.points = $\begin{bmatrix} [x_0, y_0] \\ [x_1, y_1] \\ \vdots \\ [x_{n-1}, y_{n-1}] \end{bmatrix}$ → in order entered

tri.simplices = $\begin{bmatrix} [V_i, V_j, V_K] \\ [V_l, V_m, V_n] \\ \vdots \\ [V_o, V_p, V_q] \end{bmatrix}$ → order assigned by triangulation algorithm, first vertex chosen at random, followed by #2 and #3, CCW

tri.neighbors = [[8,-1,2], [-1,5,3], ...] → "rows" are in triangle order
ordering of neighbors follows order of opposite vertices. (convenient)

Transition function:

$\delta(T_j, U_K) = T_l$ → N triangles + "-1" identifier for outside of domain:

$\delta(-1, U_K) = -1 \quad \forall U_K$ (can't leave outside once entered)
↳ No! Don't make this a state.

$U = \begin{bmatrix} 0 - \text{remain} \\ 1 - \text{depart Facet 1} \\ 2 - " \quad " \quad 2 \\ 3 - " \quad " \quad 3 \end{bmatrix}$

→ Can use "tri.neighbors" to construct the transition function structure

T.X = np.array([0,1,2,3,4,...,N-1]) ← AUTOMATIC

T.Σ = np.array([0,1,2,3,#]) ← Immutable

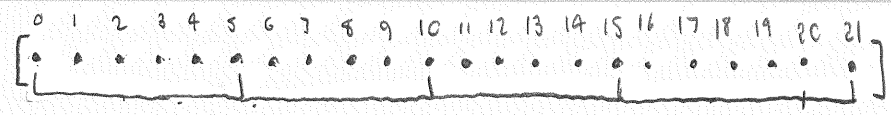
T.δ = np.array([[1,0,1],[1,1,-1],[1,2,5],...]) ← Constructed automatically from the triangulation data

T.O = np.array([[0, #], ...]) → observation map
(0 ↑ triangle ID; # ↑ label — home - 0, visit - positive integer, avoid - negative integer)

T.O = np.array([-1, 0, 1, 2]) in project
↑ set of observations
user-specified

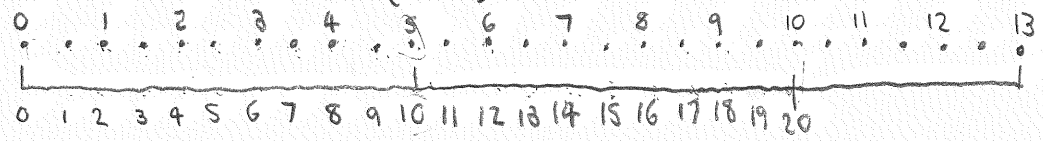
0	0	0
0	1	1
0	2	2
0	3	3
1	0	4
1	1	5
1	2	6
1	3	7
	0	8
	1	9

$4(idx1) + idx2$



dt_check = 5

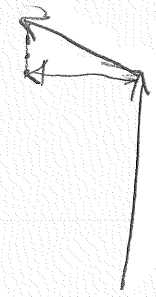
num_eval = Floor $\left(\frac{21-0}{5}\right) = 4$



N_window = (dt_check / h) + 1 → assuming these divide evenly

prev_eval_idx = 0, 1, 2

t0_local = master_time_vec (



13.058, 0.9708

```
# set options:
t0 = 0.0
tf = 100.0   ) should divide evenly
h = 0.2
dt_check = 10.0
ICs = np.array([12.5, 0.1])
traj_sched = np.array(...)
traj_dur = np.array(...)
```

compute fg-array for leaving, staying

```
for idx in range (0, traj_sched.shape[0]):
    time_temp, state_temp = integrator_stop_check (....)
    # save transition times
    # compute velocities, store?
    # step
```

Control-inside-Facet:

$\forall i,j,\ i \neq j: \hat{n}_i^T\left([A]\bar{v}_j + [B]\bar{u}_j + \bar{a}\right) \leq 0$ → This works!

→ $\hat{n}_i^T[B]\bar{u}_j \leq 0$ for path planning

$$\begin{cases} \hat{n}_1^T[B]\bar{u}_2 \leq 0 \\ \hat{n}_1^T[B]\bar{u}_3 \leq 0 \\ \hat{n}_2^T[B]\bar{u}_1 \leq 0 \\ \hat{n}_2^T[B]\bar{u}_3 \leq 0 \\ \hat{n}_3^T[B]\bar{u}_1 \leq 0 \\ \hat{n}_3^T[B]\bar{u}_2 \leq 0 \end{cases}$$

To turn around — control to desired facet. After entering, control to remain for chosen time duration. Then, control to leave out of entry facet.

* control schedule for demo: (obtaining steering-law trajectory)

T_{10}	T_9	T_8	T_0	T_8	T_9	T_{10}
U_2	U_2	U_2	$(U_S: \tau_1 \to U_1)$	U_1	U_3	$(U_S: \tau_2)$

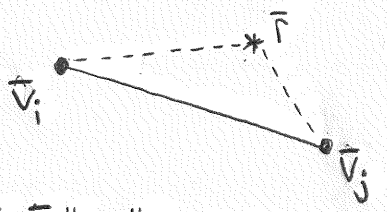
* Control types:
  - control to facet
  - control to remain
  - control to remain, then leave through a facet
* Control schedule given with expected simplex
* Use "events" option in solve_ivp to stop at transitions

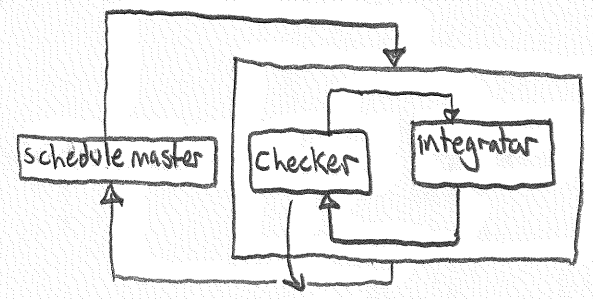
→ codes for demo should adapt readily for later simulation work.

→ Pre-computing steering-law trajectory $(\bar{r}(t), \bar{v}(t))$ and $\dot{\bar{v}}(t)$ for feedback

→ Piecewise-constant control (with sufficiently high frequency) would work best for this

Facet-crossing logic:





$$\|\bar{r} - \bar{v}_i\| + \|\bar{r} - \bar{v}_j\| - \|\bar{v}_j - \bar{v}_i\| = 0$$

* if facet cross: stop, return x, t^*, num (?), history
* does checker or schedule master handle "stay-inside then leave" cases?

```
switch_event = lambda t, x: facet_switch(t, x, tri)

def Facet_switch(t,x,tri):
    num = tri.find_simplex(x)
# Find vertices...
```

→ This won't work, but is probably a good example of using lambda function to have extra arguments in "events" function.

$LB \leq [C]\bar{u} \leq \infty$, $LB = \bar{0}_{Kx1}$

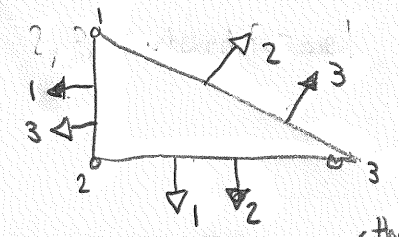
For 2: 3, 1

$\bar{0} \leq [C]\bar{u} \longrightarrow [C] =$

$$\left[\begin{array}{ccc}
-\hat{n}_2^T[B] & \bar{0}_{2x1}^T & \bar{0}_{2x1}^T \\
-\hat{n}_3^T[B] & \bar{0}_{2x1}^T & \bar{0}_{2x1}^T \\
-\hat{n}_2^T[B] - \hat{n}_3^T[B] & \bar{0}_{2x1}^T & \bar{0}_{2x1}^T \\
\hline
\bar{0}_{2x1}^T & \hat{n}_1^T[B] & \bar{0}_{2x1}^T \\
\bar{0}_{2x1}^T & -\hat{n}_3^T[B] & \bar{0}_{2x1}^T \\
\hline
\bar{0}_{2x1}^T & \bar{0}_{2x1}^T & \hat{n}_1^T[B] \\
\bar{0}_{2x1}^T & \bar{0}_{2x1}^T & -\hat{n}_2^T[B]
\end{array}\right]$$

← For 2:

$$\bar{u} = \begin{bmatrix} \bar{u}_1 \\ \bar{u}_2 \\ \bar{u}_3 \end{bmatrix}$$



$1 \to 3, \quad 2 \to 1, \quad 3 \to 2$ ← theirs

↑ mine "describes"

"My 3 is their 2"

$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{bmatrix}$$

$$\begin{bmatrix} \hat{b}^T \\ \hat{c}^T \\ \hat{a}^T \end{bmatrix}$$

$$\bar{u} = \begin{bmatrix} 0.0106863 & -0.0031337 \\ 0.0001023 & 0.0042543 \end{bmatrix} \bar{x} + \begin{pmatrix} 0.00811 \\ -0.01397 \end{pmatrix}$$

$$\dot{\bar{x}} = [F]\bar{x} + \bar{g}$$

$$\bar{x} = [\Phi_F]\bar{x}(0) + \int_0^t [\Phi_F][B]\,d\tau\,\bar{g}$$

To Do

- Code to compute fg-array for all triangles → can I add these to tri?
- Linear tracking controller
- Validate, extend LTL formulas
- product automaton, etc.

$T_0 : S_2 - \pi_3, \pi_4 \longrightarrow "\pi_2"$

$T_3 : S_1 - \pi_1, \pi_2 \longrightarrow "\pi_1"$

$T_6 : K_1 - \pi_5, \pi_6, \pi_7 \longrightarrow "\pi_4"$

$T_{10} : R - \pi_8, \pi_9 \longrightarrow "\pi_3"$

$$\dot{\bar{x}} = [A]\bar{x} + [B]\bar{u}, \quad \bar{x}_r = \begin{pmatrix} \bar{r}(t) \\ \bar{v}(t) \end{pmatrix}$$

$$\dot{\bar{r}} = [F]\bar{r} + \bar{g} \longrightarrow \bar{r}(t) = [\Phi_F(t,t_0)]\bar{r}_0 + \int_0^t [\Phi_F(\tau,0)]\,d\tau\,\bar{g}$$

$$\bar{v}(t) = [F]\bar{r}(t) + \bar{g}$$

$$\dot{\bar{r}} = \bar{v} = e^{-\alpha(t-t^*)}\left([F]_a\bar{r} + \bar{g}_a\right) + \left(1 - e^{-\alpha(t-t^*)}\right)\left([F]_b\bar{r} + \bar{g}_b\right) \text{ — with blending}$$

$$\bar{e} = \bar{x} - \begin{pmatrix} \bar{r} \\ z^* \\ \bar{v} \\ 0 \end{pmatrix}, \quad \dot{\bar{e}} = [A]\bar{x} + [B]\bar{u} - \begin{pmatrix} \dot{\bar{r}} \\ 0 \\ \dot{\bar{v}} \\ 0 \end{pmatrix}, \quad \dot{\bar{z}} = \begin{pmatrix} \bar{v} \\ 0 \\ \dot{\bar{v}} \\ 0 \end{pmatrix}$$

$$\dot{\bar{e}} = [A]\left(\bar{e} + \begin{pmatrix} \bar{r} \\ z^* \\ \bar{v} \\ 0 \end{pmatrix}\right) + [B]\bar{u} - \begin{pmatrix} \dot{\bar{r}} \\ 0 \\ \dot{\bar{v}} \\ 0 \end{pmatrix}$$

$$\dot{\bar{e}} = [A]\bar{e} + [A]\bar{z} + [B]\bar{u} - \dot{\bar{z}} \qquad [A]\bar{z} = \left[\begin{array}{ccc|ccc} 0&0&0&1&0&0 \\ 0&0&0&0&1&0 \\ 0&0&0&0&0&1 \\ \hline 0&0&0&0&0&0 \\ 0&0&0&0&0&0 \\ 0&0&0&0&0&0 \end{array}\right]\begin{pmatrix} \bar{r} \\ z^* \\ \bar{v} \\ 0 \end{pmatrix} = \begin{pmatrix} \bar{v} \\ 0 \\ \hline \bar{0}_{3\times1} \end{pmatrix}$$

$$\bar{u} = -[K]\bar{e} + \ldots$$

$$[A]\bar{z} - \dot{\bar{z}} = \begin{pmatrix} \bar{0}_{3\times1} \\ \hline -\dot{\bar{v}} \\ 0 \end{pmatrix}$$

$$[B]\bar{u} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ g & 0 \\ 0 & -g \\ 0 & 0 \end{bmatrix}\begin{pmatrix} \theta \\ \phi \end{pmatrix} = -[B][K]\bar{e} + \begin{pmatrix} \bar{0}_{3\times1} \\ \hline \dot{\bar{v}} \\ 0 \end{pmatrix}$$

Just ignore $\bar{z}$ for now... $\bar{e} = \bar{x} - \begin{pmatrix} \bar{r} \\ \bar{v} \end{pmatrix}$, $\dot{\bar{e}} = [A]\left(\bar{e} + \begin{pmatrix} \bar{r} \\ \bar{v} \end{pmatrix}\right) + [B]\bar{u} - \begin{pmatrix} \bar{v} \\ \dot{\bar{v}} \end{pmatrix}$

$$\dot{\bar{e}} = [A]\bar{e} + [A]\bar{z} + [B]\bar{u} - \dot{\bar{z}}, \quad [A]\bar{z} - \dot{\bar{z}} = \begin{pmatrix} \bar{0}_{2\times1} \\ \hline -\dot{\bar{v}} \end{pmatrix}$$

$$\bar{u} = -[K]\bar{e} + \ldots$$

$$[B]\bar{u} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ g & 0 \\ 0 & -g \end{bmatrix}\begin{pmatrix} \theta \\ \phi \end{pmatrix} = -\underset{4\times2}{[B]}\overset{2\times4}{[K]}\bar{e} + \begin{pmatrix} \bar{0}_{2\times1} \\ \hline \dot{\bar{v}} \end{pmatrix}$$

$$\left|\lambda[I] - ([A] - [B][K])\right| \rightarrow \text{choose } [K] \text{ to enforce negative real part for eigenvalues}$$

For control with discrete update

For idx in range(len(time_vec)):

→ control_now = control_Fcn(mode_vec[idx], states..., constants, OESO_LMO, OESO_GMO, t[idx])

→ integrate dynamics for one step with this control held constant, save state values

→ Codes are general enough to enable mode and control computation inside the ODE function

constants. u_const = control_now

To Do 4/12:

- complete project tasks 8-11 for control with discrete update, test (write a new .py file for this?)

→ time_vec = np.linspace(t0, tF, N)

for N = 11, $t_0 = 0$, $t_F = 1$:

time_vec = 0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0

N = len(time_vec) = floor($(t_F - t_0)/h$) + 1

thus $N = \text{floor}\left(\frac{t_F - t_0}{h}\right) + 1, \quad 0 \le r < 1$

$$\frac{(t_F - t_0)}{N-1} = \text{Floor}\left(\frac{t_F - t_0}{N-1}\right) + r$$

$$\text{so } r = \frac{t_F - t_0}{h} - \text{Floor}\left(\frac{t_F - t_0}{h}\right) = \text{np.fmod}\left((t_F - t_0), h\right)$$

$$h \cdot N = \frac{t_F - t_0}{N-1} \cdot h + 1 - r \rightarrow N - 1 + r = \frac{t_F - t_0}{h}$$

$$h = \frac{t_F - t_0}{N - 1 + r} \; ? \quad \ldots.$$

→ Choose h, t_F, t_0 such that $\frac{t_F - t_0}{h}$ divides evenly.

Then, $idx(t^*) = \frac{t^* - t_0}{h}$

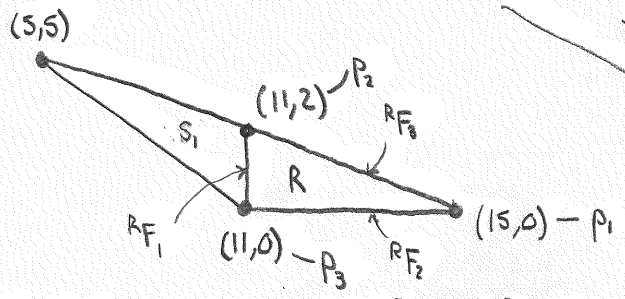
→ For task 8, with h = 0.2, passed t = 15s and t = 400s, but not 100, 200

When set h = 1.0, all checks passed!

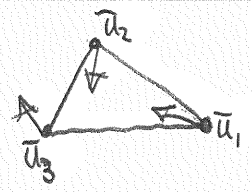
→ For task 9, with h = 1.0, passed all but t = 400s. There's an MRP switch in the MRP error $\sigma_{B/R}$ not long after 200s. Reduce to h = 0.2, passed!

→ For task 10, with = 0.2, passed t = 15s and t = 400s.

When set h = 1.0, pass checks for t = 100s and t = 200s!



$$\min_{u_1,u_2} \left(1 - u_1^2 - u_2^2\right)^2 \quad \text{s.t. } \left(1 - \|\vec{u}\|^2\right)^2$$



$$\min_{\vec{u}_1,\vec{u}_2,\vec{u}_3} \left(C - \|\vec{u}_1\| - \|\vec{u}_2\| - \|\vec{u}_3\|\right)^2$$

subject to ....

↓

scipy.optimize.minimize()

Facet-based control out of left face of R, then top face of S_1:

$$\hat{n}_3 = \frac{(-4,2)^T}{\|(-4,2)\|}, \quad \hat{n}_2 = \begin{pmatrix}0\\-1\end{pmatrix}, \quad [B] = \begin{bmatrix}1&0\\0&1\end{bmatrix}, \quad [A] = \begin{bmatrix}0&0\\0&0\end{bmatrix}$$

$$\hat{n}_3 = \begin{pmatrix}-0.89442719\\0.44721136\end{pmatrix}$$

p_1: $\hat{n}_2^T[B]\vec{u}_{(1)} \le 0 : (0,-1)\begin{pmatrix}u_{(1)1}\\u_{(1)2}\end{pmatrix} \le 0 \rightarrow u_{(1)2} \ge 0$

$\hat{n}_3^T[B]\vec{u}_{(1)} \le 0 : \left(-4/2\sqrt{5}, 2/2\sqrt{5}\right)\begin{pmatrix}u_{(1)1}\\u_{(1)2}\end{pmatrix} \le 0 \rightarrow \left(\frac{-2}{\sqrt{5}}, \frac{1}{\sqrt{5}}\right)\begin{pmatrix}u_{(1)1}\\u_{(1)2}\end{pmatrix} \le 0$

$\rightarrow u_{(1)2} - 2u_{(1)1} \le 0$

$\sqrt{16+4}$ $\hat{n}_3 = \begin{bmatrix}0&1\\-1&0\end{bmatrix}\begin{pmatrix}-2/\sqrt{5}\\1/\sqrt{5}\end{pmatrix} = \begin{pmatrix}1/\sqrt{5}\\2/\sqrt{5}\end{pmatrix} \rightarrow u_{(1)1} + 2u_{(1)2} \le 0$

$\sqrt{20}$

$$u_{(1)2} \le -\frac{1}{2}u_{(1)1} \quad \text{or} \quad u_{(1)1} \le -2u_{(1)2}$$

$2\sqrt{5}$ $\hat{n}_2^T[B]\vec{u}_{(1)} + \hat{n}_3^T[B]\vec{u}_{(1)} < 0 \rightarrow (0,-1)\begin{pmatrix}u_{(1)1}\\u_{(1)2}\end{pmatrix} + \left(\frac{-2}{\sqrt{5}}, \frac{2}{\sqrt{5}}\right)\begin{pmatrix}u_{(1)1}\\u_{(1)2}\end{pmatrix} < 0$

$-u_{(1)2} + \frac{1}{\sqrt{5}}u_{(1)1} + \frac{2}{\sqrt{5}}u_{(1)2} < 0$

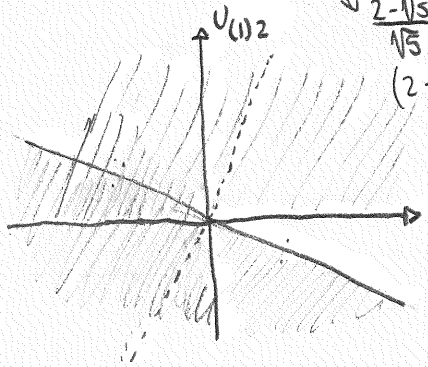
$\frac{\sqrt{5}}{\sqrt{5}}$ $\frac{-\sqrt{5}+1}{\sqrt{5}}u_{(1)2} + \frac{1}{\sqrt{5}}u_{(1)1} < 0 \rightarrow -2u_{(1)1} < -(1-\sqrt{5})u_{(1)2}$

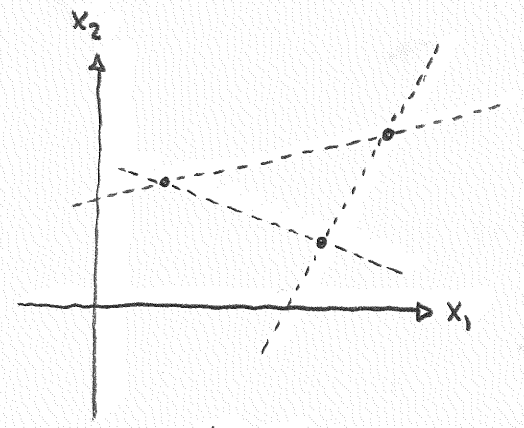
$\frac{2-\sqrt{5}}{\sqrt{5}}u_{(1)2} + \frac{1}{\sqrt{5}}u_{(1)1} < 0$ $\qquad -2u_{(1)1} < (\sqrt{5}-1)u_{(1)2}$

$(2-\sqrt{5})u_{(1)2} + u_{(1)1} < 0$ $\qquad u_{(1)1} > \frac{-(\sqrt{5}-1)}{2}u_{(1)2}$

$(\sqrt{5}-2)u_{(1)2} > u_{(1)1}$

$u_{(1)2} > \frac{1}{\sqrt{5}-2}u_{(1)1}$





$$m_k = \frac{y_j - y_i}{x_j - x_i}, \quad y - y_i = m_k(x - x_i), \quad j \neq k,\ i \neq k,\ j \neq i$$

$$y = y_i + m_k(x - x_i)$$

→ For a given (x, y), test with all (or a subset) inequalities. Should be fast for small graphs.

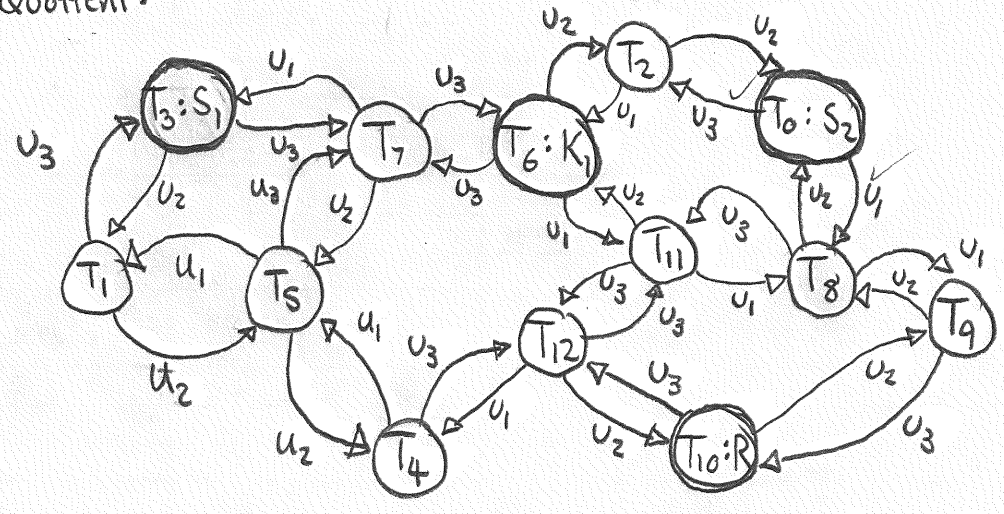
→ Based on current position and overall objective, control to facet.

Use "tri.Find_simplex(np.array([x, y]))" to determine which triangle number the point (x, y) is in.

Smooth the control at transitions from T_a to T_b:

$$\bar{u}(t) = e^{-\alpha(t-t^*)}\bar{u}_a(t) + \left(1 - e^{-\alpha(t-t^*)}\right)\bar{u}_b(t)$$

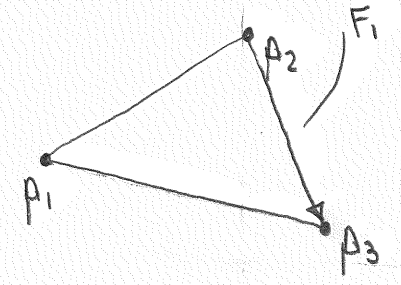
Quotient:



(Exit transitions not shown)

checked all transitions ✓

where $u_i \rightarrow$ "leave through facet i", $i = 1, 2, 3$



$\bar{r}_{ij} = \bar{p}_j - \bar{p}_i$, $\bar{p}_i = \begin{pmatrix} x_i \\ y_i \end{pmatrix}$

$[R] = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$ → rotate right 90°

$\bar{r}_{23} = \begin{pmatrix} 2 \\ 1 \end{pmatrix} - \begin{pmatrix} 0 \\ 6 \end{pmatrix} = \begin{pmatrix} 2 \\ -5 \end{pmatrix}$

$[R] = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$ → rotate left 90°

$[R]\bar{r}_{23} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}\begin{pmatrix} 2 \\ -5 \end{pmatrix} = \begin{pmatrix} -5 \\ -2 \end{pmatrix}$, choose j, i for ccw vector → always rotate right!

$\dot{\bar{x}} = \underbrace{\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}\bar{x}}_{\text{disturbance dynamics?}} + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\bar{u}$, ← velocity → $\bar{u}$ commands act as steering law since they are velocity

---

To leave facet F_1 when $[A] = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$:

$$\left.\begin{aligned} &\hat{n}_2^T[B]\bar{u}_{(1)} \le 0 \\ &\hat{n}_3^T[B]\bar{u}_{(1)} \le 0 \\ &\hat{n}_2^T[B]\bar{u}_{(1)} + \hat{n}_3^T[B]\bar{u}_{(1)} < 0 \end{aligned}\right\} \text{at vertex } p_1$$

For vertex p_2: $\hat{n}_1^T[B]\bar{u}_{(2)} > 0$, $\hat{n}_3^T[B]\bar{u}_{(2)} \le 0$

For vertex p_3: $\hat{n}_1^T[B]\bar{u}_{(3)} > 0$, $\hat{n}_2^T[B]\bar{u}_{(3)} \le 0$

→ Partition domain into triangles, do system design for first-order system. This gives target $\bar{r}(t), \bar{v}(t)$ for control. PID?

Linearized quadcopter dynamics:

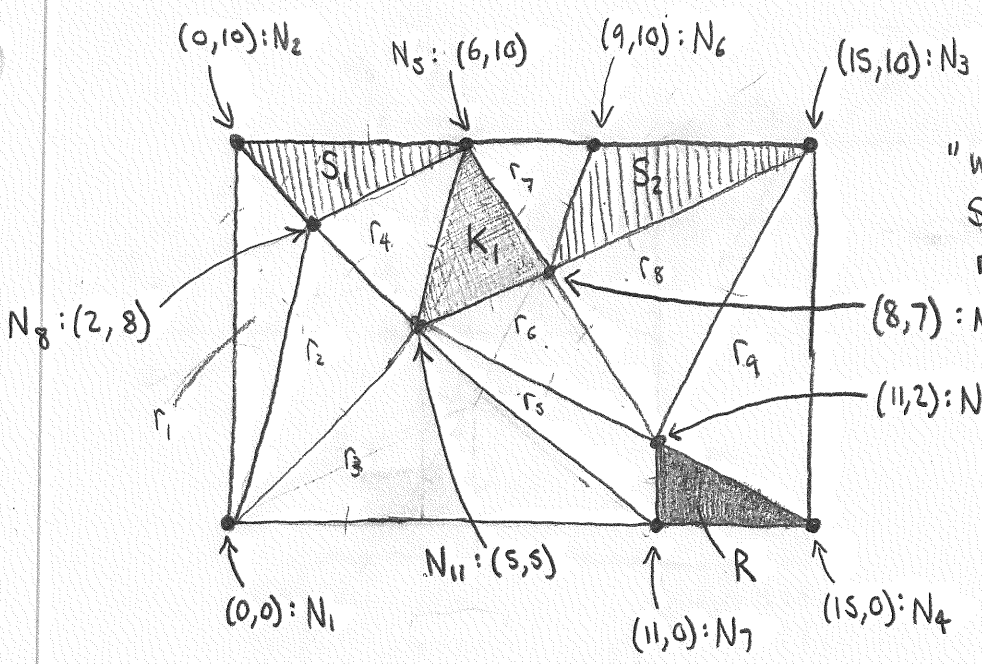
$\ddot{X} = g\theta$

$\ddot{Y} = -g\phi$

$\ddot{Z} = 0$

To Do:

- Test control to facet of triangle
- Build LTL formula, automaton



always
"while avoiding K_1, visit S_1 then S_2 and return to R, repeat forever"

Domain is partitioned into nodes, whose connections form simplices.
Nodes define vertices of bounding triangles of important regions.

- With linearized dynamics, $\dot{\bar{x}} = [A]\bar{x} + [B]\bar{u} + \bar{a}$, where [A] could have added "wind vector field" component (?)
- Step 1: define all polytopes in domain mathematically:
  - node positions ✓ → 11 nodes
  - predicates & normal vectors
- Step 2: Implement control to vertices → making these regions invariants
- Step 3: Construct LTL formula φ and automaton A_φ, build "quotient" of domain
- Step 4: Build product automaton, obtain control strategy
- Step 6: Implement control to satisfy φ
- Step 7: Add realism to above control, dynamics
- Step 8: Generalizations, next steps, explore complexities

r_1: [(0,0), (0,10), (2,8)], edges w/ r_2, S_1, end.
r_2: [(0,0), (2,8), (5,5)], edges w/ r_1, r_3, r_4
r_3: [(0,0), (11,0), (5,5)], edges w/ r_2, r_5, end
r_4: [(5,5), (2,8), (6,10)], edges w/ r_2, K_1, S_1
r_5: [(11,0), (5,5), (11,2)], edges w/ r_3, r_6, R
r_6: [(11,2), (5,5), (8,7)], edges w/ r_5, r_8, K_1
r_7: [(8,7), (9,10), (6,10)], edges w/ K_1, S_2, end
r_8: [(11,2), (15,10), (8,7)], edges w/ r_6, r_9, S_2
r_9: [(15,0), (11,2), (15,10)], edges w/ r_8, R, end
K_1: [(5,5), (8,7), (6,10)], edges w/ r_4, r_6, r_7
S_1: [(2,8), (6,10), (0,10)], edges w/ r_1, r_4, end
S_2: [(8,7), (15,10), (9,10)], edges w/ r_7, r_9, end
R: [(11,0), (15,0), (11,2)], edges w/ r_5, r_9, end