9. ANEXO I. Uso de Fnyzer y resultados.

A continuación, se muestra el modelo realizado con fnyzer de la conexión de dos secciones de forma directa, sin semáforos ni intersecciones, como lo explicado en la Sección 4. Modelo de un sistema de tráfico con redes flexibles.

Este modelo se ha creado como un diccionario de Python de red flexible, aplicando la estructura definida de la librería de fnyzer. En este diccionario se definen los lugares, las transiciones y los manejadores de eventos y de intensidades con sus desigualdades y ecuaciones. Luego, las características relativas al muestreo con la técnica MPC y se escribe la función objetivo que se quiere alcanzar, esto es, minimizar el marcado de P1.

```
section_two_mpc = {
    'name': 'section_two_mpc',
    'solver': 'gurobi',
    'places': {'P1': 45, 'P2': 1},
    'trans': {'T0': {'l0': 0, 'a0': 0}, 'T1': {'l0': 0, 'a0': 0}, 'T2': {'l0': 0, 'a0':
0}},
    'vhandlers': {
        'v0': [{'a': ('T0', 'v0'), 'x1': ('v0', 'P1')}, 'a == x1'],
        'v1': [{'y1': ('P1', 'v1'), 'z1': ('T1', 'v1'), 'x2': ('v1', 'P2')}, 'y1 == z1',
'y1 == x2'],
        'v2': [{'y2': ('P2', 'v2'), 'z2': ('T2', 'v2')}, 'y2 == z2']
    },
    'regs': {
        'freeflow1': ["m['P1'] <= " + str(endff)],
        'constant1': [str(endff) + " <= m['P1']", "m['P1'] <= " + str(endcf)],
        'stuck1': [str(endcf) + " <= m['P1']"],

        'freeflow2': ["m['P2']<= " + str(endff)],
        'constant2': [str(endff) + " <= m['P2']", "m['P2'] <= " + str(endcf)],
    },
    'parts': {'Part1': ['freeflow1', 'constant1', 'stuck1'],
              'Part2': ['freeflow2', 'constant2']},
    'shandlers': {
        's1': [
            {'b1': ('P1', 's1'), 'c1': ('s1', 'T1'), 'b2': ('P2', 's1'), 'c2': ('s1',
'T2')},
            {'freeflow1': ["c1 == (b1)*" + str(ff1)],
            'constant1': ["c1 == " + str(cf1)],
            'stuck1': ["c1 == " + str(endsf) + "+" + str(stuckslope1) + "*(b2)"],
            'freeflow2': ["c2 == b2*" + str(ff1)],
            'constant2': ["c2 == " + str(cf1)],
            }],

    },
    # 'mbounds': ["m['P1']>=0"],
    'obj': {'f': "m['P1']", 'sense': 'min'},
    'W': 100,
    'wl': 0,
    'wu': 500,
    'options': {
        'antype': 'mpc',
        'mpc': {
            'firstinlen': 0.3,
            'numsteps': 10,
            'maxnumins': 1,
            'flexins': False
        },
        'writevars': {'m': ['P1', 'P2'], 'L': 'all', 'U': 'all'},
        'plotres': True,
        'plotvars': {'evm': ['P1', 'P2']},
        # 'xlsfile': 'seccion.xls',
        'printres': False,
    },
    'actfplaces': 'all',
    'exftrans': 'all',
    'actavplaces': 'all',
    'exavtrans': 'all',
}
```

10. ANEXO II. Modelo programado de un sistema de tráfico

El código referente a lo que se explica a continuación está subido a un repositorio de GitHub para poder ser consultado de manera libre.

La clase *SectionLight* contiene los atributos característicos de una sección de tráfico, esto es, el vector del marcado y del flujo más las constantes inherentes al comportamiento del sistema, q, h, r, maxcar. Además, posee los atributos del semáforo, un vector que almacena todos los estados anteriores del semáforo, α y β y un valor boolean que dice si esa sección tiene o no semáforo.

La unión de estas secciones será lo que conforme nuestra carretera/modelo de tráfico. Estas secciones se almacenarán en un vector que se crea de la siguiente forma:

```
hsections = [Tml.SectionLight(15, True, 3, 3), Tml.SectionLight(10, False, 0, 0)]
```

Se ha creado un vector con dos secciones conectadas.

Se han definido muchos métodos que se pueden encontrar en el repositorio que se ha nombrado anteriormente, voy a explicar los más relevantes:

Calculo del flujo:

Esta función pasa por cada sección del vector *Section*. Primero, se calcula el flujo que tiene la sección según la ecuación 4, posteriormente, si la sección tiene semáforo, se aplica el factor corrector: en rojo flujo=0, en ambar se reduce la velocidad.

```
def new_flow_horizontal_mpc(section, i, lambd, delta, sequence, m):
    for j in range(0, len(section)):
        if j < len(section) - 1:
            f = lambd[j] * min(section[j].m[i+m] / section[j].q, section[j].h, (sec-
tion[j].maxcar - section[j+1].m[i+m]) / section[j+1].r)
            if section[j].tl:
                if sequence[m] == 1:
                    f = f * section[j].alfa / (2 * delta)
                elif sequence[m] == 2:
                    f = 0
                elif sequence[m] == 3:
                    f = f * section[j].beta / (2 * delta)
            section[j].f.append(f)
        else:
            f = lambd[j] * min(section[j].m[i+m] / section[j].q, section[j].h)
            if section[j].tl:
                if sequence[m] == 1:
                    f = f * section[j].alfa / (2 * delta)
                elif sequence[m] == 2:
                    f = 0
                elif sequence[m] == 3:
                    f = f * section[j].beta / (2 * delta)
            section[j].f.append(f)
```

Calculo del marcado:

El marcado se calcula según la ecuación 13. Sobre el vector *Section,* se coge el marcado actual, se resta los coches que se van y se suman los que entran, en el periodo delta.

```
def new_marking_horizontal_mpc(section, inputcars, i, delta, m):
    for j in range(0, len(section)):
        if j == 0:
            section[j].m.append(section[j].m[i+m] + delta * inputcars[m] - delta * sec-
tion[j].f[i+m])
        else:
            section[j].m.append(section[j].m[i+m] + delta * section[j-1].f[i+m] - delta
* section[j].f[i+m])
```

Máquina de estados:

Esta función solamente es útil cuando no se aplica ninguna medida correctiva, esto es, que la secuencia de activación de los semáforos está predeterminada.

Según el estado actual del semáforo, se calcula el próximo estado en función de los periodos que le quedan en ese estado. Por ejemplo, si tiene que estar en verde 4 periodos y lleva 2, el próximo estado será verde.

```
def state_machine(section, i):
    if section.lstate[i] == 0:
        section.cnt += 1
        if section.cnt == section.gg:
            section.cnt = 0
            section.lstate.append(1)
        else:
            section.lstate.append(0)
    if section.lstate[i] == 1:
        section.lstate.append(2)
    if section.lstate[i] == 2:
        section.cnt += 1
        if section.cnt == section.rr:
            section.cnt = 0
            section.lstate.append(3)
        else:
            section.lstate.append(2)
    if section.lstate[i] == 3:
        section.lstate.append(0)
```

Invertir máquina de estados:

Hay una pequeña función para asignar a la otra dirección de la intersección la máquina de estados inversa. Si en la dirección horizontal esta en verde, a la dirección perpendicular se le asignará el estado rojo.

```
def invert_nextstate(i):
    j = 0
    if i == 0:
        j = 2
    elif i == 1:
        j = 3
    elif i == 2:
        j = 0
    elif i == 3:
        j = 1
    return j
```

Control MPC - algoritmo:

Como se ha descrito en la Sección 5. Programación de un sistema de tráfico, se ha implementado la técnica de control MPC:

- Estado actual = Estado inicial.

```
"""CURRENT STATE = INITIAL STATE"""
if i < samples:
    inputV = Tml.inputcars(vinputcars, i)
inputH = Tml.inputcars(hinputcars, i)
sectionH = copy.deepcopy(hsections)
sectionV = copy.deepcopy(vsections)
state = hsections[0].lstate[i-1]
next_state = 0
```

- Bucle:

Calcular los resultados de las combinaciones posibles, guardar la combinación que dé el máximo flujo de salida

```
for j in range(0, len(sequenceH)):
# Iterate among the possible 8 combinations to find the maximum ec 12.
    for m in range(0, len(sequenceH[0])):  # Iteration for the next 4 states
        """FLOW"""
        Tml.new_flow_horizontal_mpc(sectionH, i, hlambda, delta, sequenceH[j], m)
        Tml.new_flow_vertical_mpc(sectionV, i, vlambda, delta, sequenceV[j], m)

        """MARKING"""
        Tml.new_marking_horizontal_mpc(sectionH, inputH, i, delta, m)
        Tml.new_marking_vertical_mpc(sectionV, inputV, i, delta, m)

    """GET FLOW"""
    # appends the flow for each combination so later I know the index of the maxi-
mum flow combination
    flow.append(sectionH[1].f[i + 3] + sectionV[0][1].f[i + 3])
    Tml.remove(sectionH, sectionV)

"""GET THE INDEX FOR THE MAXIMUM FLOW COMBINATION"""
maxflow = max(flow)
index = 0
for j in range(0, len(sequenceH)):
    if flow[j] == maxflow: index = j

"""GET THE FIRST STATE OF THAT SEQUENCE"""
next_state = sequenceH[index][0]
```

Por último, aplicar la primera acción del nuevo estado

```
"""APPLY NEXT STATE"""
hsections[0].lstate.append(next_state)
vsections[0][0].lstate.append(Tml.invert_nextstate(next_state))

"""CALCULATE NEW FLOW"""
Tml.new_flow_horizontal_blind(hsections, i, hlambda, delta)
Tml.new_flow_vertical_blind(vsections, i, vlambda, delta)

"""CALCULATE NEW MARKING"""
Tml.new_marking_horizontal_blind(hsections, hinputcars, i, delta)
if i < samples/2:
    Tml.new_marking_vertical_blind(vsections, vinputcars, i, delta)
else:
    Tml.new_marking_vertical_blind(vsections, vinputcars2, i, delta)
```