

Aprendizaje Profundo para la Manipulación Robótica Adaptativa en Escenarios Complejos



Universidad
Internacional
de Valencia

Autor: Jesús Pedro Gallego Berciano

Directora: Yaneth Coromoto Moreno Caldera

Máster Universitario en Inteligencia Artificial

De:
 Planeta Formación y Universidades

Contenido

- INTRODUCCIÓN
 - a. Objetivos
 - b. Justificación
 - c. Alcance
- ESTADO DEL ARTE
- MATERIALES
 - a. Hardware
 - b. Software
 - c. Dataset
 - d. Redes Neuronales
- MÉTODOLÓGÍA
- DESARROLLO DEL PROYECTO
- CONCLUSIONES Y LÍNEAS FUTURAS
- VIDEO SIMULACIÓN

Introducción

Objetivos

Optimizar algoritmos de aprendizaje profundo para mejorar la precisión, eficiencia y adaptabilidad de la manipulación robótica en entornos no estructurados, permitiendo a los robots ajustar sus acciones en tiempo real según las condiciones cambiantes del entorno y las características diversas de los objetos a manipular.

1. Identificar las limitaciones actuales en la manipulación robótica en entornos no estructurados.
2. Desarrollar algoritmos de aprendizaje profundo adaptados para la manipulación robótica.
3. Crear un entorno de simulación que represente condiciones no estructuradas y variaciones de objetos.
4. Establecer métricas claras para evaluar la precisión, eficiencia y adaptabilidad de los algoritmos.
5. Realizar pruebas en el entorno de simulación para evaluar estas métricas.

Justificación

- ❖ El avance en IA y robótica móvil ha aumentado la demanda de aplicaciones robóticas en entornos no estructurados y cambiantes.
- ❖ Es necesario desarrollar nuevos sistemas de control que mejoren la fiabilidad y eficiencia de los robots, especialmente en aplicaciones de agarre robótico.
- ❖ Los robots tradicionales están diseñados para tareas específicas, lo cual limita su adaptabilidad y capacidad de ampliación.
- ❖ El uso de sistemas de agarre avanzados se ha expandido a sectores como:
 - Medicina
 - Logística
 - Automoción
 - Robótica colaborativa

Alcance

- ❖ Estudio y desarrollo de algoritmos de aprendizaje profundo aplicados al agarre robótico, adaptados para la manipulación de piezas en un entorno industrial.
- ❖ Para el entrenamiento y evaluación de estos algoritmos se usará la plataforma de simulación robótica IsaacSim.
- ❖ El uso de esta plataforma permitirá la creación de entornos que representen condiciones no estructuradas y con variedad de objetos

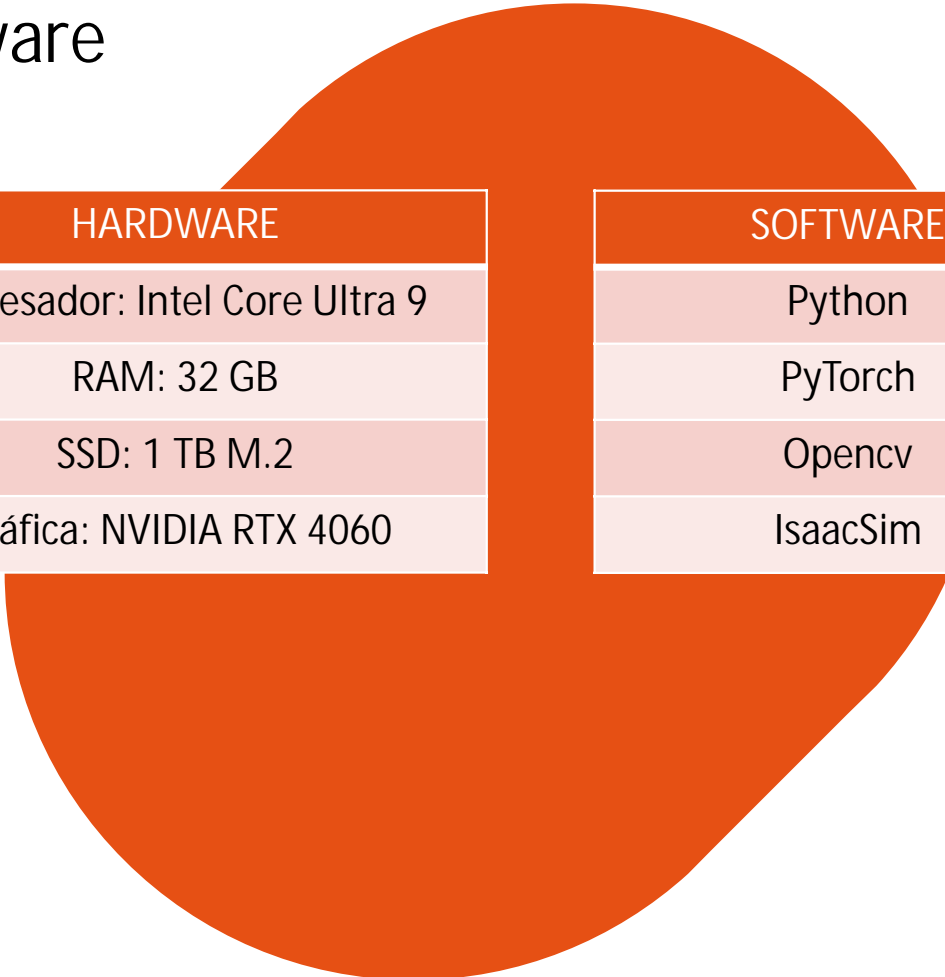
Estado del Arte

Estado del Arte

AUTOR	TITULO	CONTENIDO
Sun et al.	<u>Research challenges and progress in robotic grasping and manipulation competitions.</u>	Desafíos en el agarre robótico
Kleeberger et al.	<u>A survey on learning-based robotic grasping.</u>	Clasificación de los distintos enfoques en agarre robótico
Lobbezoo y Kwon	<u>Simulated and real robotic reach, grasp, and pick-and-place using combined reinforcement learning and traditional controls.</u>	Simulación robótica aplicada al agarre robótico y al pick and place
Sharma et al.	<u>Digital twins: State of the art theory and practice, challenges, and open research questions.</u>	Estado del arte de los gemelos industriales

Materiales

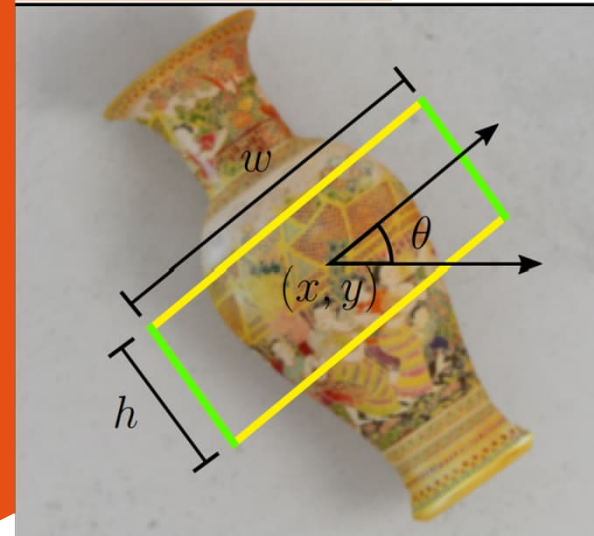
Hardware y Software



HARDWARE	SOFTWARE
Procesador: Intel Core Ultra 9	Python
RAM: 32 GB	PyTorch
SSD: 1 TB M.2	Opencv
Gráfica: NVIDIA RTX 4060	IsaacSim

Dataset

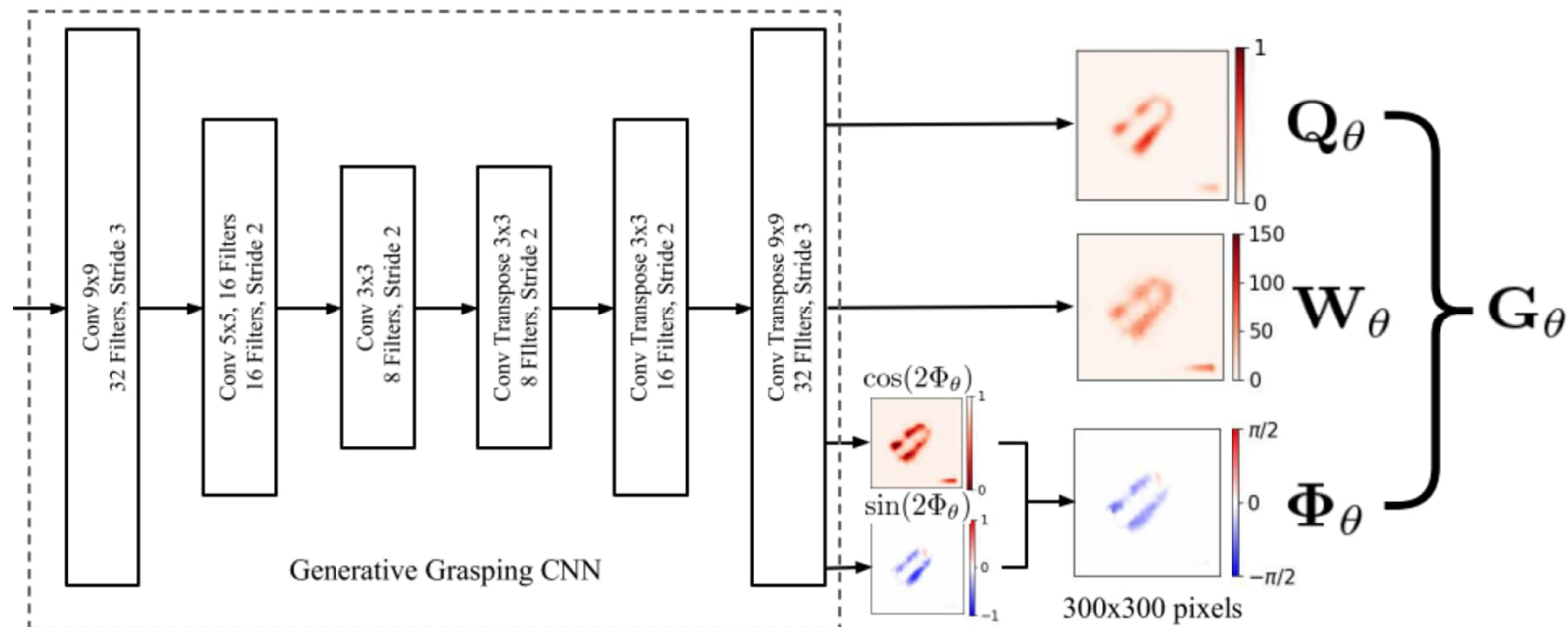
[Depierre et al. \(2018\)](#) presentan Jacquard, un conjunto de datos sintéticos a gran escala. Jacquard contiene más de 1.1 millones de posiciones de agarre anotadas en 11,000 objetos diferentes, proporcionando imágenes RGB-D realistas con etiquetas de agarre verificadas mediante simulación.



Redes neuronales

Generative Grasping Convolutional Neuronal Network (GGCNN)

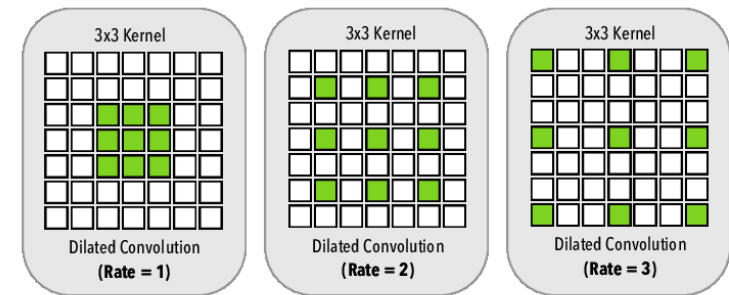
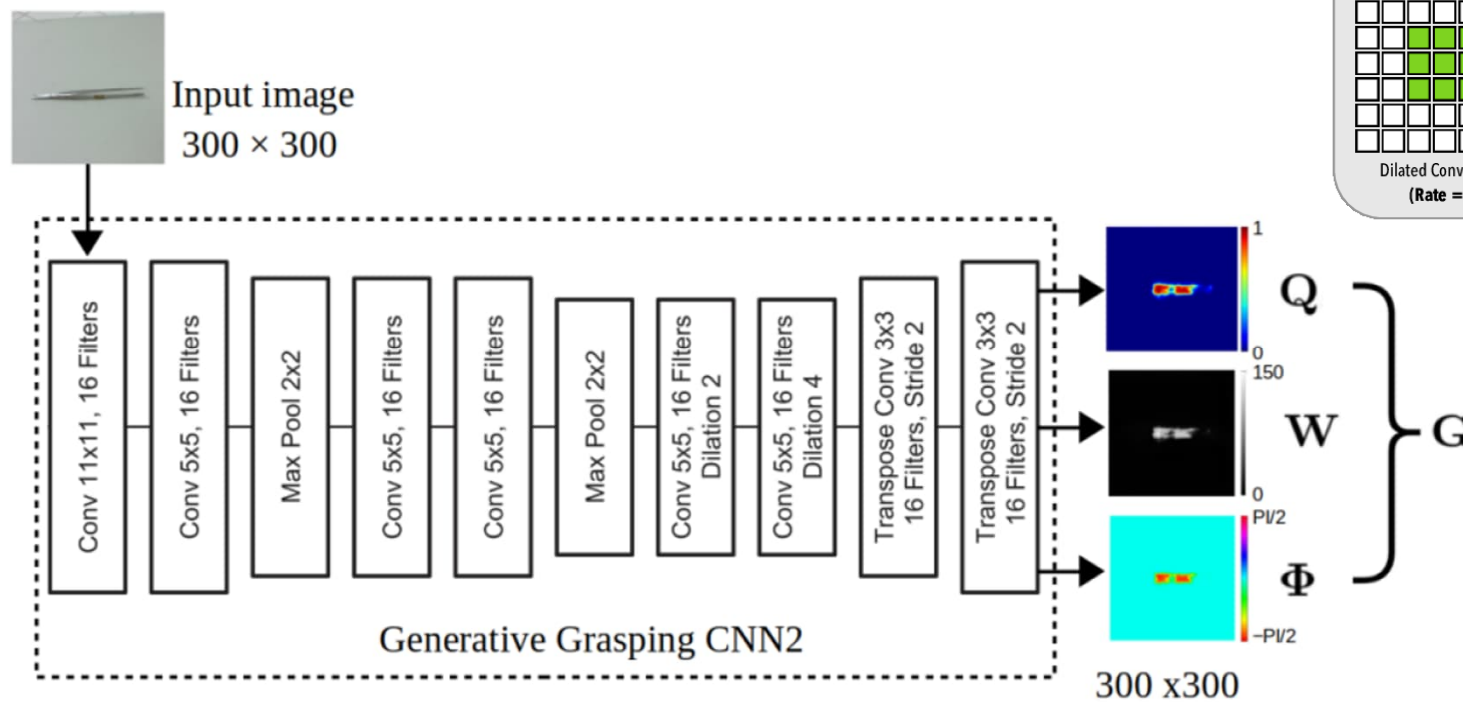
[Morrison et al. \(2018\)](#)



Redes neuronales

Generative Grasping Convolutional Neuronal Network 2 (GGCNN2)

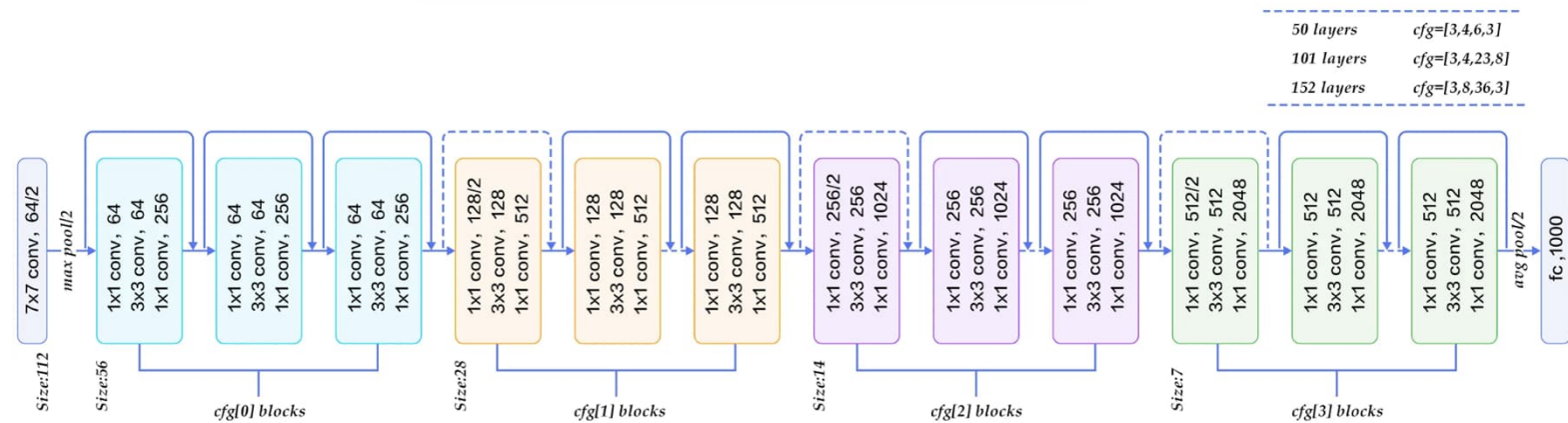
[Morrison et al. \(2018\)](#)



Redes neuronales

Residual Network (ResNet)

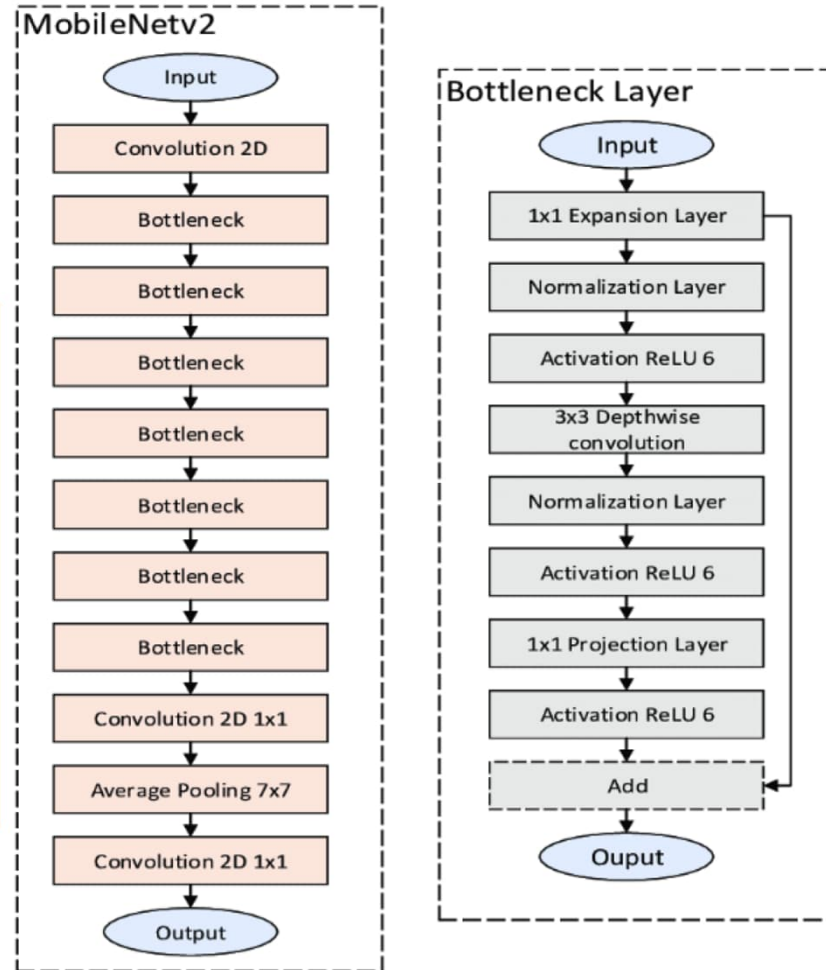
[He et al. \(2015\)](#)



Nombre Red	Arquitectura
ResNet-18	18 capas
ResNet-34	34 capas
ResNet-50	50 capas
ResNet-101	101 capas
ResNet-152	152 capas

Redes neuronales MobileNetV2

[Sandler et al. \(2018\)](#)



Redes neuronales

Comparativa entre redes

GGCNN vs GGCNN2

GGCNN2 mejora a GGCNN en términos de precisión, eficiencia y generalización.

Usa convoluciones con dilatación y una mejor arquitectura para generar mapas de agarre más precisos.

Es más rápida y requiere menos recursos computacionales que GGCNN

ResNet-50 vs MobileNetV2

ResNet-50 es una red profunda con bloques residuales, ideal para extracción de características en tareas complejas, pero es muy pesada para aplicaciones en robots en tiempo real.

MobileNetV2 está diseñada para ser ligera y eficiente, por lo que es más adecuada para robots móviles con restricciones de hardware.

GGCNN2 vs ResNet-50 y MobileNetV2

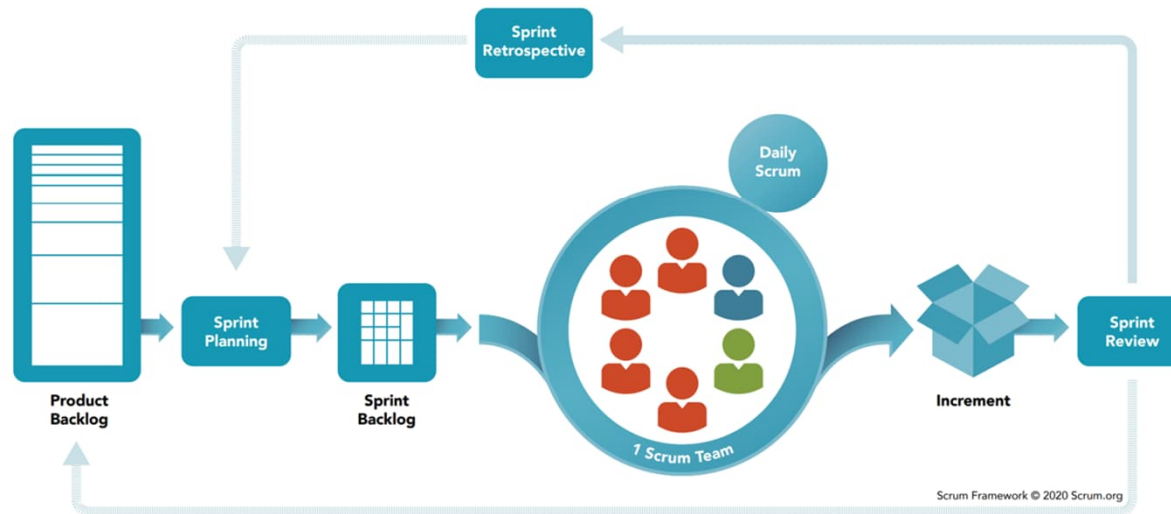
GGCNN2 es específica para predicción de agarres, mientras que ResNet-50 y MobileNetV2 son redes generales para visión por computadora.

Si se necesita velocidad en tiempo real, GGCNN2 o MobileNetV2.

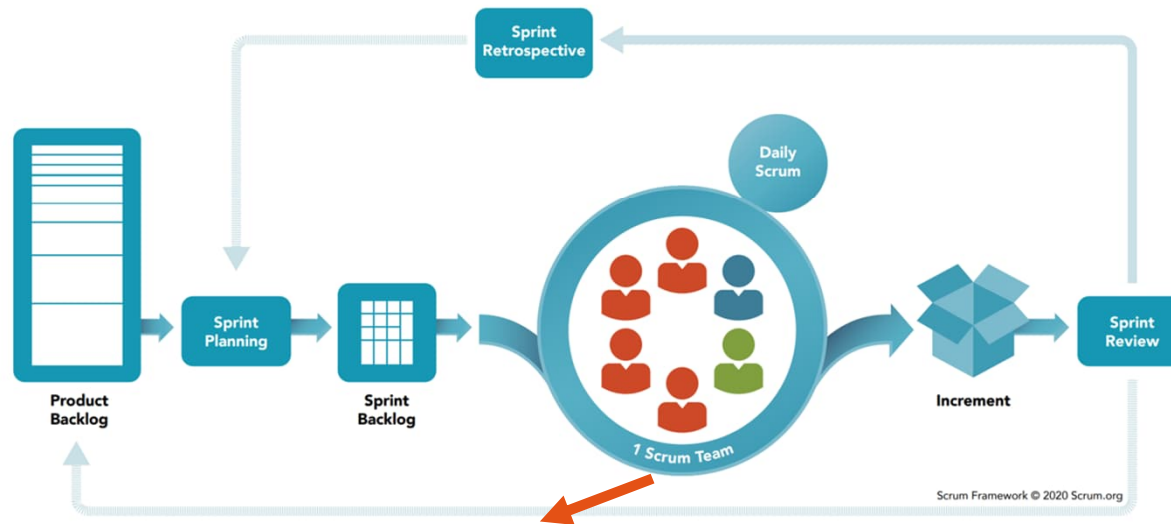
Si se busca alta precisión sin importar el tiempo de inferencia, ResNet-50

Metodología

SCRUM FRAMEWORK



SCRUM FRAMEWORK

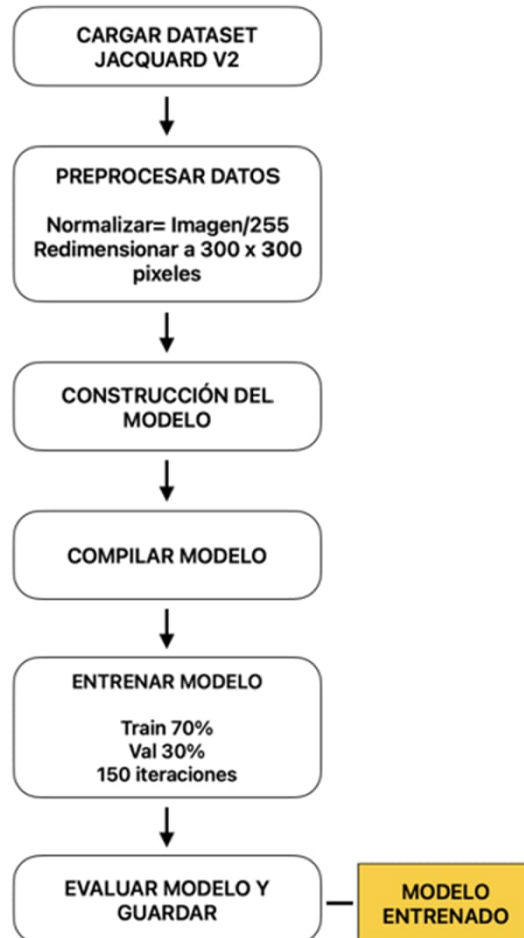


Rol en el equipo	Tarea a realizar	Adaptación
Scrum Master	Facilita el proceso, elimina obstáculos y asegura que se sigan las prácticas de Scrum.	Encargándome de optimizar el proceso y resolver bloqueos.
Product Owner	Representa los intereses del cliente y prioriza los requisitos en el Product Backlog.	Definiendo los objetivos, requisitos y prioridades del proyecto.
Development Team	Grupo multidisciplinar que desarrolla el producto y entrega incrementos funcionales en cada sprint.	Realizando las tareas de investigación, desarrollo, experimentación y redacción.

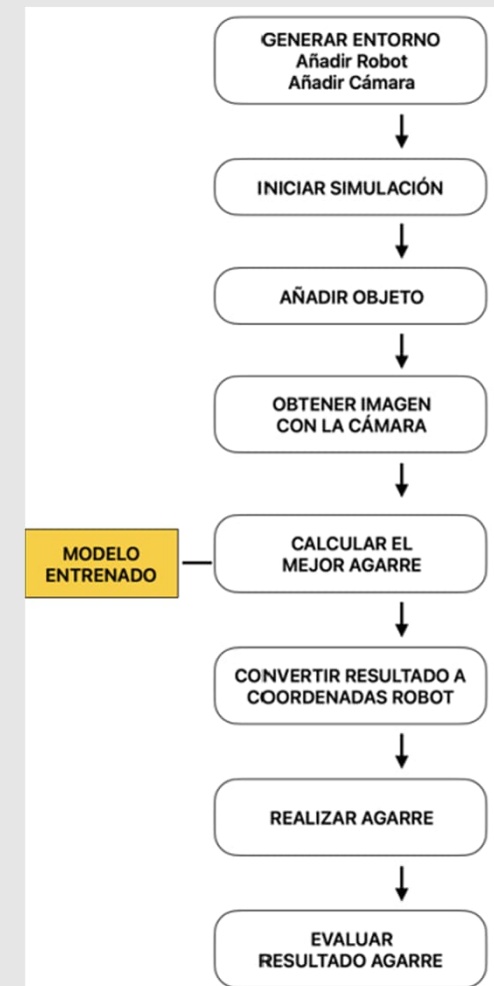
Evento	Tareas a realizar
Creación Product Backlog	<p>Revisión bibliográfica sobre aprendizaje profundo y manipulación robótica</p> <p>Implementación y prueba de modelos de aprendizaje profundo</p> <p>Análisis de resultados y optimización de los algoritmos</p> <p>Redacción y revisión de los distintos capítulos del documento.</p>
Planificación de Sprints	<p>Inicio del sprint: definición de objetivos y selección de tareas del backlog.</p> <p>Trabajo diario: desarrollo de tareas y autoevaluaciones rápidas.</p> <p>Reunión con el tutor: presentación de avances y retroalimentación.</p> <p>Revisión y retrospectiva: reflexión sobre lo logrado y ajustes para el siguiente sprint.</p>
Reuniones con el tutor como Sprint Review	<p>Presenté los avances realizados.</p> <p>Recibí feedback y sugerencias de mejora.</p> <p>Ajusté el backlog y redefiní las prioridades según la retroalimentación recibida.</p>
Reflexión y Mejora Continua	<p>¿Qué aspectos del trabajo han funcionado bien?</p> <p>¿Cuáles fueron los principales bloqueos o dificultades?</p> <p>¿Cómo podría mejorar la organización del próximo sprint?</p>

Desarrollo

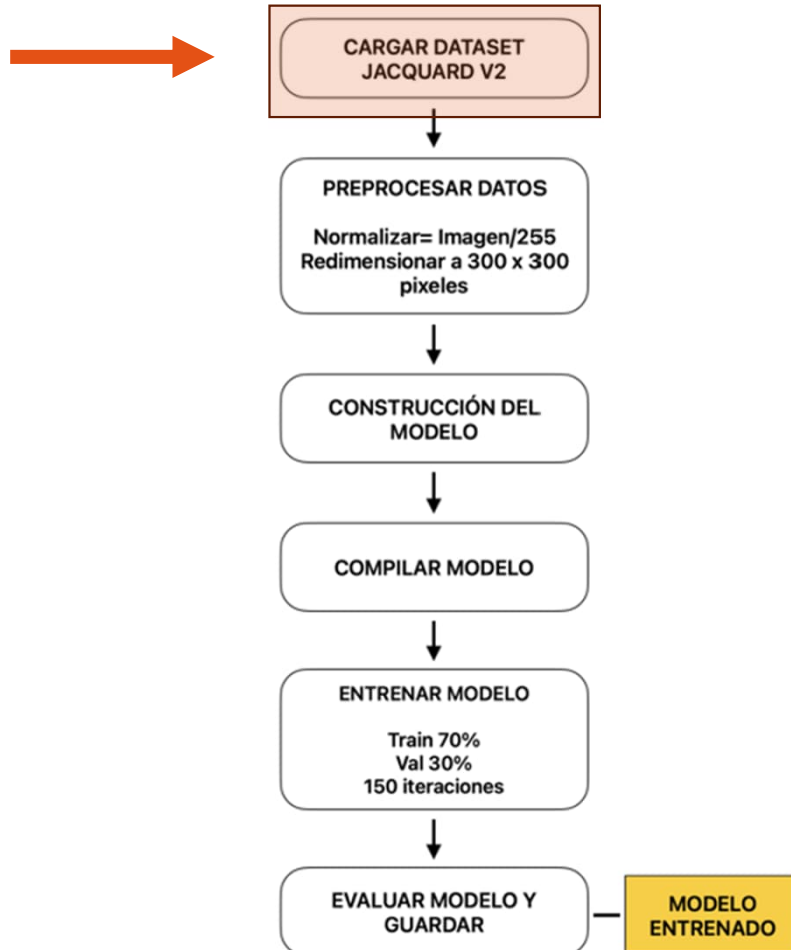
Entrenamiento



Simulación



Entrenamiento

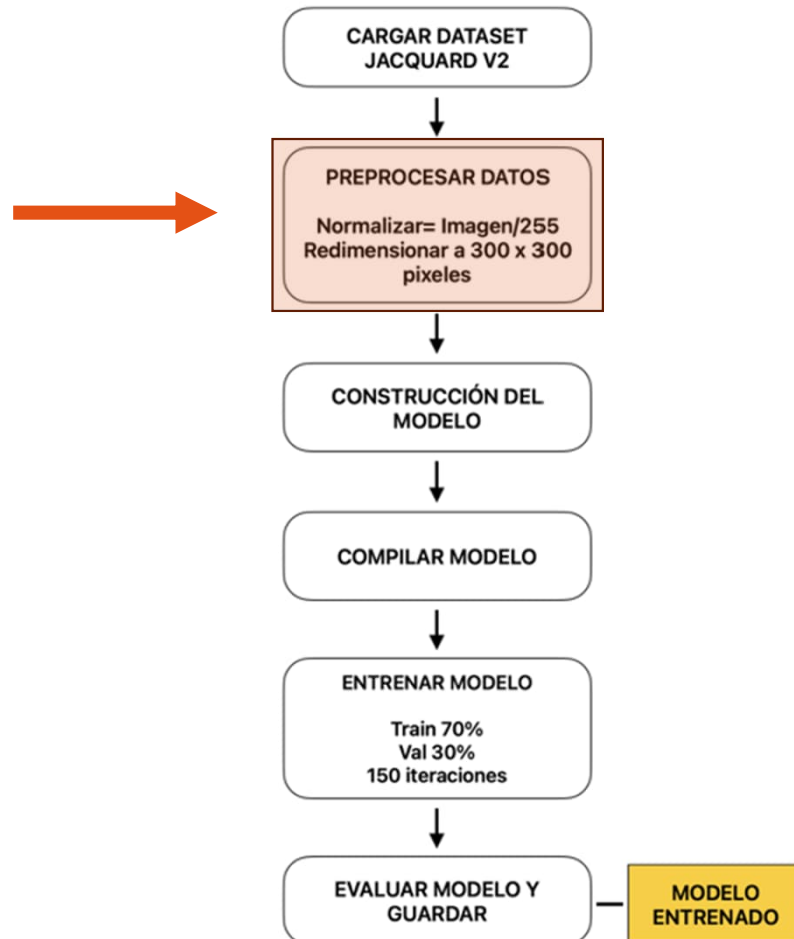


```
# Load Dataset
logging.info('Loading {} Dataset...'.format(args.dataset.title()))
Dataset = get_dataset(args.dataset)

train_dataset = Dataset(args.dataset_path, start=0.0, end=args.split, ds_rotate=args.ds_rotate,
                        random_rotate=True, random_zoom=True,
                        include_depth=args.use_depth, include_rgb=args.use_rgb)
train_data = torch.utils.data.DataLoader(
    train_dataset,
    batch_size=args.batch_size,
    shuffle=True,
    num_workers=args.num_workers
)

val_dataset = Dataset(args.dataset_path, start=args.split, end=1.0, ds_rotate=args.ds_rotate,
                    random_rotate=True, random_zoom=True,
                    include_depth=args.use_depth, include_rgb=args.use_rgb)
val_data = torch.utils.data.DataLoader(
    val_dataset,
    batch_size=1,
    shuffle=False,
    num_workers=args.num_workers
)
```

Entrenamiento



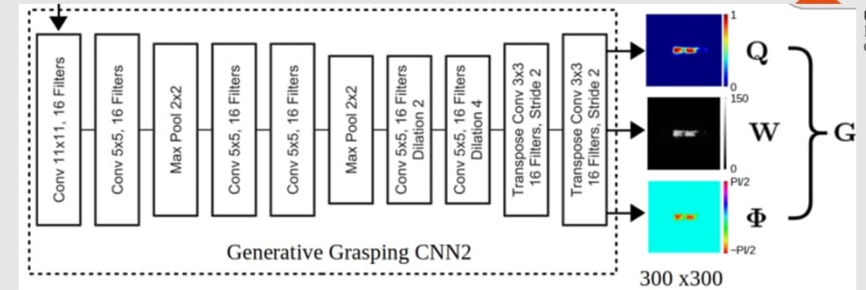
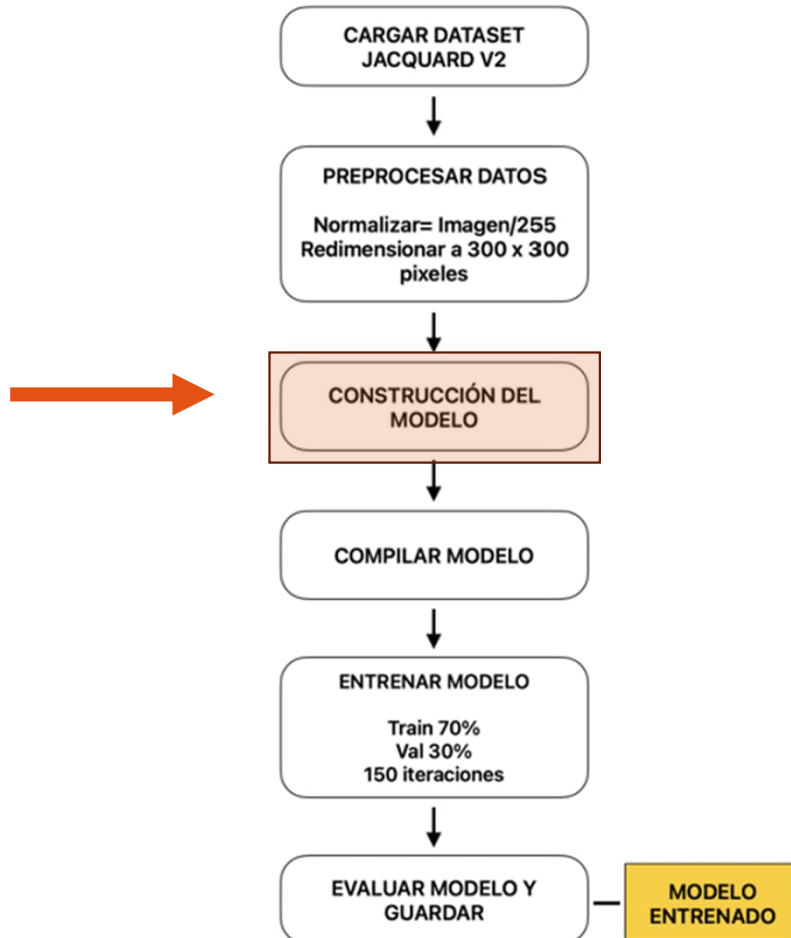
```

def get_gtbb(self, idx, rot=0, zoom=1.0):
    gtbbbs = grasp.GraspRectangles.load_from_jacquard_file(self.grasp_files[idx], scale=self.output_size / 1024.0)
    c = self.output_size//2
    gtbbbs.rotate(rot, (c, c))
    gtbbbs.zoom(zoom, (c, c))
    return gtbbbs

def get_depth(self, idx, rot=0, zoom=1.0):
    depth_img = image.DepthImage.from_tiff(self.depth_files[idx])
    depth_img.rotate(rot)
    depth_img.normalise()
    depth_img.zoom(zoom)
    depth_img.resize((self.output_size, self.output_size))
    return depth_img.img

def get_rgb(self, idx, rot=0, zoom=1.0, normalise=True):
    rgb_img = image.Image.from_file(self.rgb_files[idx])
    rgb_img.rotate(rot)
    rgb_img.zoom(zoom)
    rgb_img.resize((self.output_size, self.output_size))
    if normalise:
        rgb_img.normalise()
        rgb_img.img = rgb_img.img.transpose((2, 0, 1))
    return rgb_img.img
  
```


Entrenamiento



```

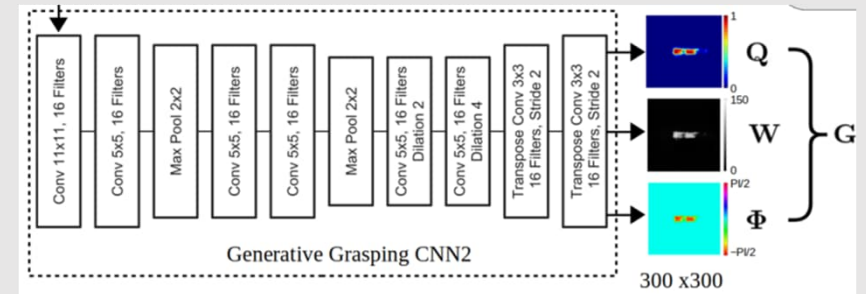
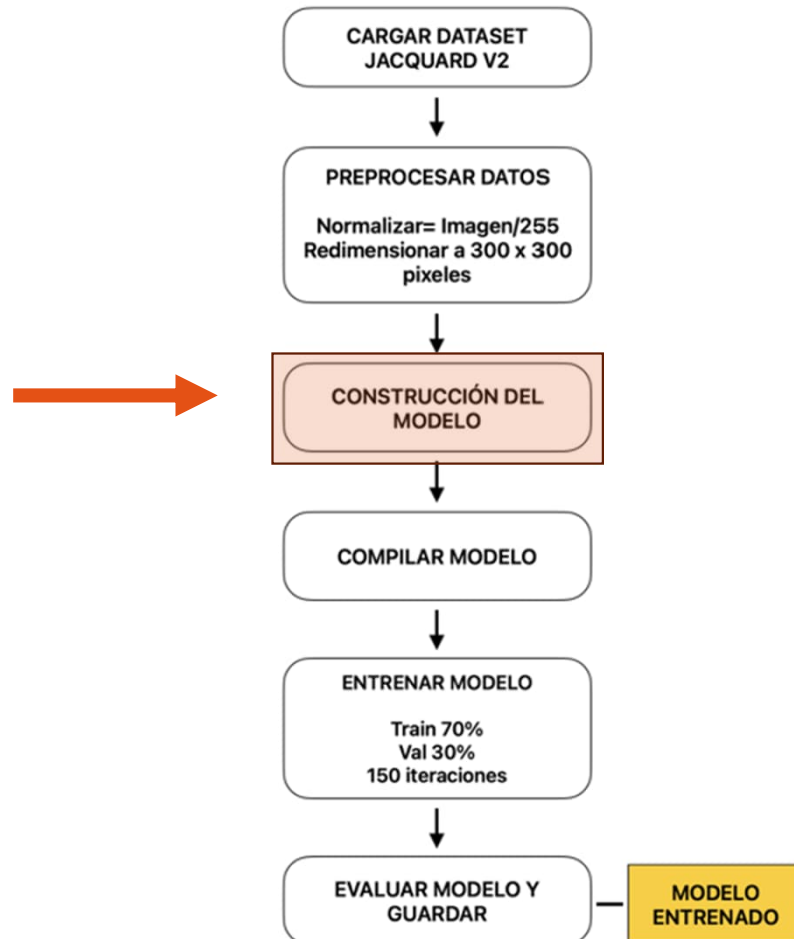
self.features = nn.Sequential(
    # 4 conv layers.
    nn.Conv2d(input_channels, filter_sizes[0], kernel_size=11, stride=1, padding=5, bias=True),
    nn.ReLU(inplace=True),
    nn.Conv2d(filter_sizes[0], filter_sizes[0], kernel_size=5, stride=1, padding=2, bias=True),
    nn.ReLU(inplace=True),
    nn.MaxPool2d(kernel_size=2, stride=2),

    nn.Conv2d(filter_sizes[0], filter_sizes[1], kernel_size=5, stride=1, padding=2, bias=True),
    nn.ReLU(inplace=True),
    nn.Conv2d(filter_sizes[1], filter_sizes[1], kernel_size=5, stride=1, padding=2, bias=True),
    nn.ReLU(inplace=True),
    nn.MaxPool2d(kernel_size=2, stride=2),

    # Dilated convolutions.
    nn.Conv2d(filter_sizes[1], filter_sizes[2], kernel_size=dil_k_size, dilation=dilations[0],
              stride=1, padding=(dil_k_size//2 * dilations[0]), bias=True),
    nn.ReLU(inplace=True),
    nn.Conv2d(filter_sizes[2], filter_sizes[2], kernel_size=dil_k_size, dilation=dilations[1],
              stride=1, padding=(dil_k_size//2 * dilations[1]), bias=True),
    nn.ReLU(inplace=True),

    # Output layers
    nn.UpsamplingBilinear2d(scale_factor=2),
    nn.Conv2d(filter_sizes[2], filter_sizes[3], 3, padding=1),
    nn.ReLU(inplace=True),
    nn.UpsamplingBilinear2d(scale_factor=2),
    nn.Conv2d(filter_sizes[3], filter_sizes[3], 3, padding=1),
    nn.ReLU(inplace=True),
)
  
```

Entrenamiento



```

def forward(self, x):
    x = self.features(x)

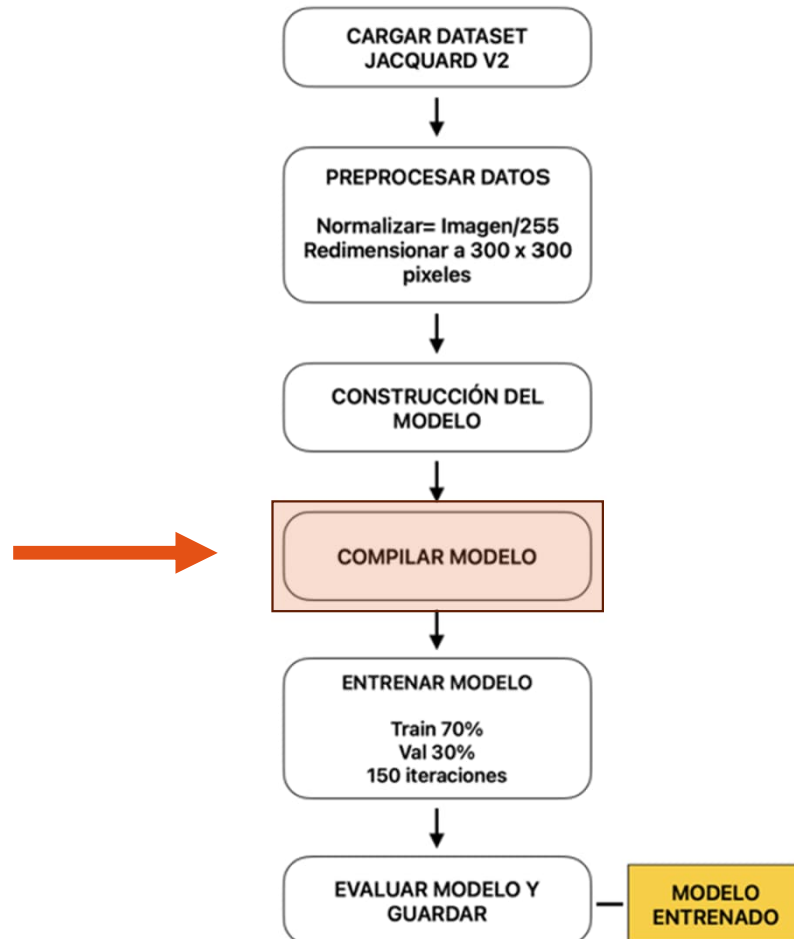
    pos_output = self.pos_output(x)
    cos_output = self.cos_output(x)
    sin_output = self.sin_output(x)
    width_output = self.width_output(x)

    return pos_output, cos_output, sin_output, width_output

def compute_loss(self, xc, yc):
    y_pos, y_cos, y_sin, y_width = yc
    pos_pred, cos_pred, sin_pred, width_pred = self(xc)

    p_loss = F.mse_loss(pos_pred, y_pos)
    cos_loss = F.mse_loss(cos_pred, y_cos)
    sin_loss = F.mse_loss(sin_pred, y_sin)
    width_loss = F.mse_loss(width_pred, y_width)
  
```

Entrenamiento



```

# Load the network
logging.info('Loading Network...')
input_channels = 1*args.use_depth + 3*args.use_rgb

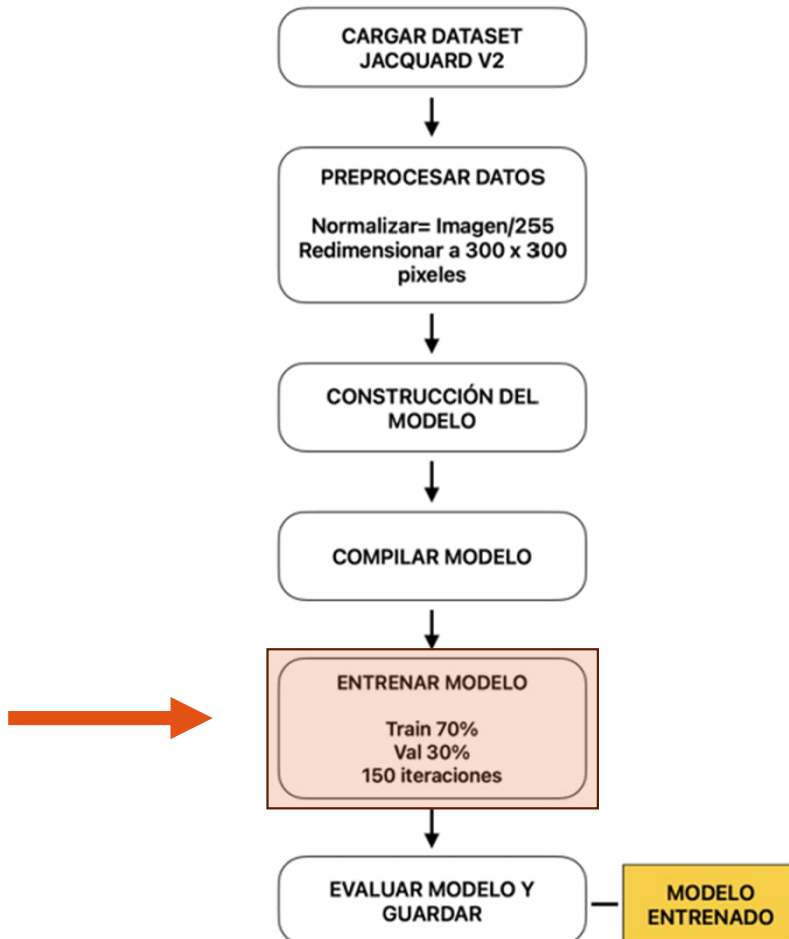
mobilenet = get_network(args.network)
net = mobilenet(input_channels=input_channels) #modify

net = net.to(device)
optimizer = optim.Adam(net.parameters(), lr=0.0005) #0.00025;0.0005;0.001
logging.info('Done')
  
```

```

def get_network(network_name):
    network_name = network_name.lower()
    if network_name == 'ggcnn':
        from .ggcnn import GGCNN
        return GGCNN
    elif network_name == 'ggcnn2':
        from .ggcnn2 import GGCNN2
        return GGCNN2
    elif network_name == 'mobilev2':
        from .mobilenetv2 import MobileNetV2
        return MobileNetV2
    elif network_name == 'resnet50':
        from .resnet import resnet50
        return resnet50
    else:
        raise NotImplementedError('Network {} is not implemented'.format(network_name))
  
```

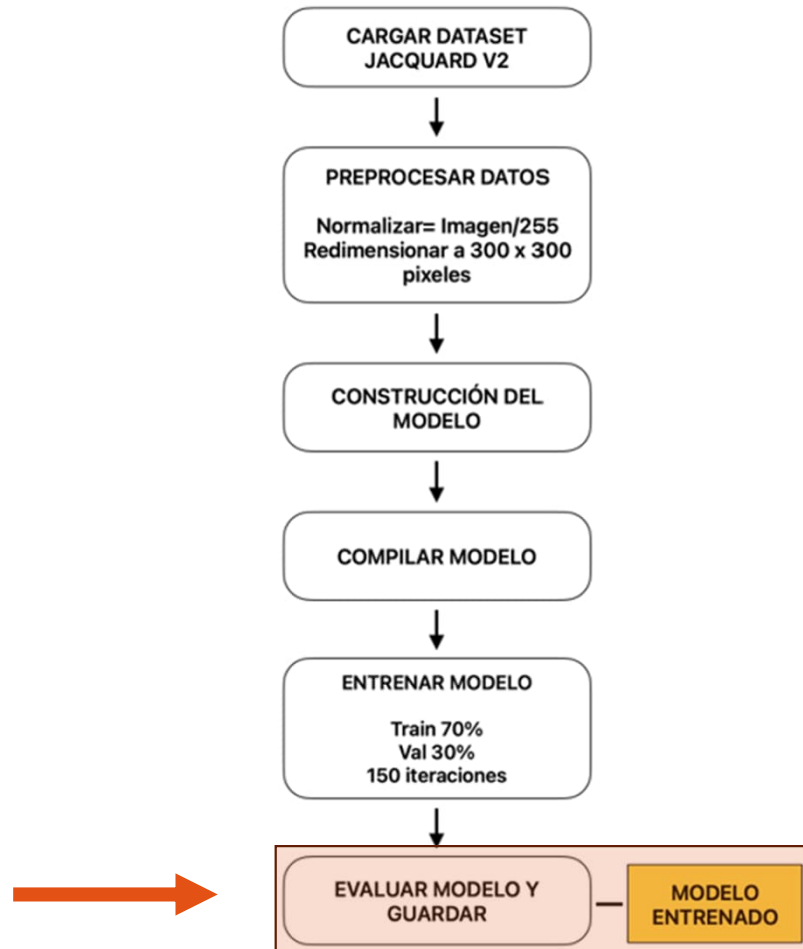
Entrenamiento



```
for epoch in range(args.epochs):
    logging.info('Beginning Epoch {:02d}'.format(epoch))
    train_results = train(epoch, net, device, train_data, optimizer, args.batches_per_epoch, vis=args.vis)
    Epoch.append(epoch)
    # Log training losses to tensorboard
    tb.add_scalar('loss/train_loss', train_results['loss'], epoch)
    for n, l in train_results['losses'].items():
        tb.add_scalar('train_loss/' + n, l, epoch)

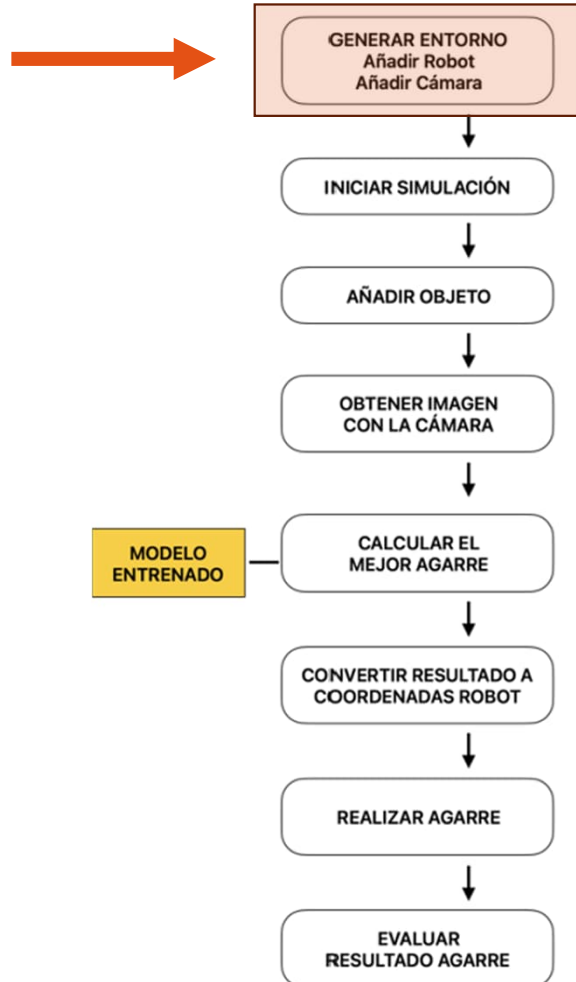
    # Run Validation
    logging.info('Validating...')
    test_results = validate(net, device, val_data, args.val_batches)
    logging.info('%d/%d = %f' % (test_results['correct'], test_results['correct'] + test_results['failed'],
                                test_results['correct']/(test_results['correct']+test_results['failed'])))
    with open('train_value.txt', 'a') as f:
        f.write(str(test_results['correct']/(test_results['correct']+test_results['failed'])))
        f.write("\n")
```

Entrenamiento



```
# Save best performing network
iou = test_results['correct'] / (test_results['correct'] + test_results['failed'])
# scheduler.step(iou)
Loss.append(train_results['loss'])
IOU.append(iou)
with open(save_folder + '/IOU_value.txt', 'a') as f:
    f.write(str(IOU))
    f.write("\n")
with open(save_folder + '/Loss_value.txt', 'a') as f:
    f.write(str(Loss))
    f.write("\n")
if iou > best_iou or epoch == 0 or (epoch % 10) == 0:
    torch.save(net, os.path.join(save_folder, 'epoch_%02d_iou_%.2f' % (epoch, iou)))
    torch.save(net.state_dict(), os.path.join(save_folder, 'epoch_%02d_iou_%.2f_loss_%.2f_statedict.pt' % (epoch, iou, train_results['loss'])))
    best_iou = iou
```

Simulación

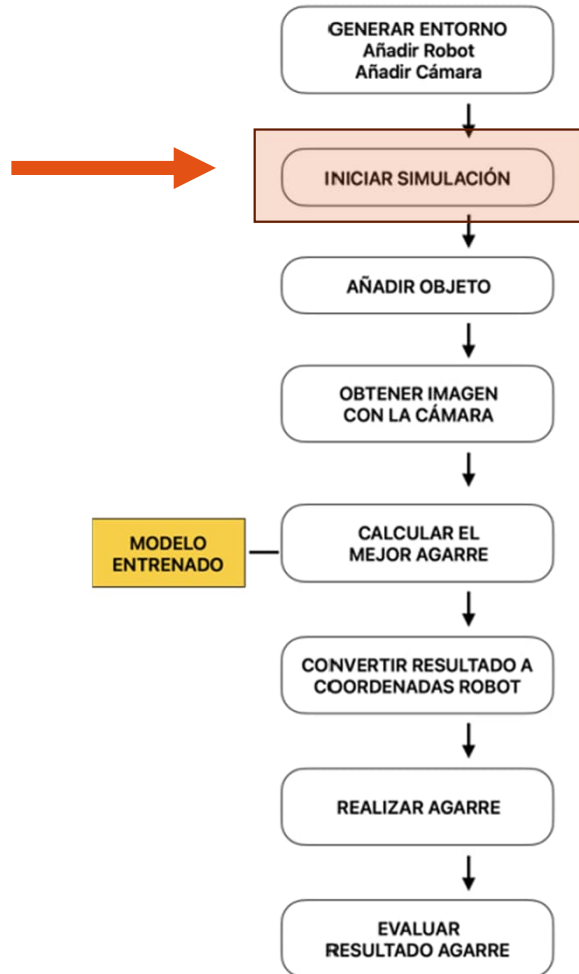


```
my_franka = my_world.scene.add(
    SingleManipulator(
        prim_path="/World/Franka", name="my_franka", end_effector_prim_name="panda_leftfinger", gripper=gripper
    )
)

camera = Camera(
    prim_path="/World/camera",
    position=np.array([0.46, 0.54, 2.0]), # Posicion (x, y, z)
    orientation=rot_utils.euler_angles_to_quats(np.array([0, 90, 0]), degrees=True), # Orientacion (cuaternion)
    frequency=20.0, # Frecuencia de captura en Hz
    #resolution=(1024, 1024)
    resolution=(300, 300)
) # Resolucion de la camara (ancho xalto)

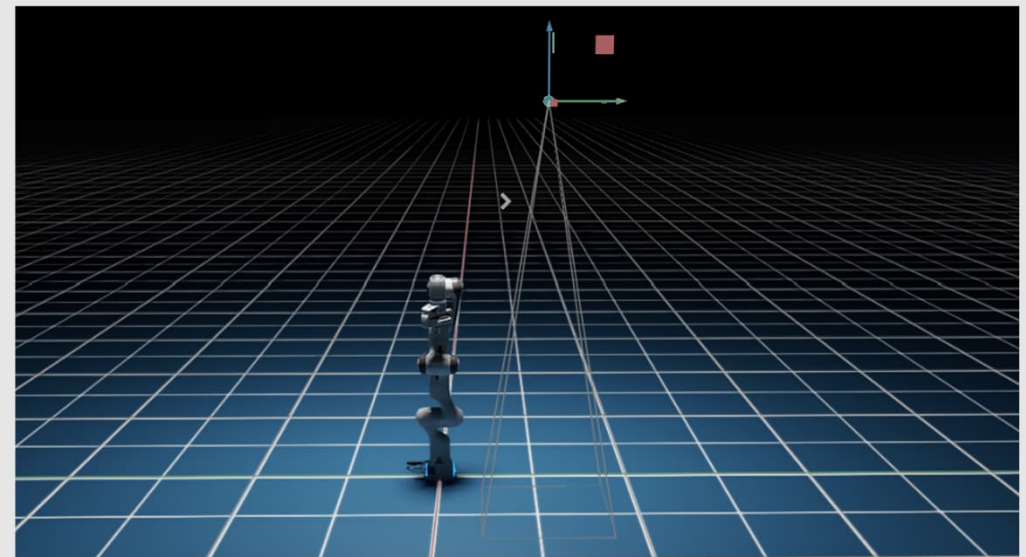
# Anadir la camara al mundo
my_world.scene.add(camera)
```


Simulación

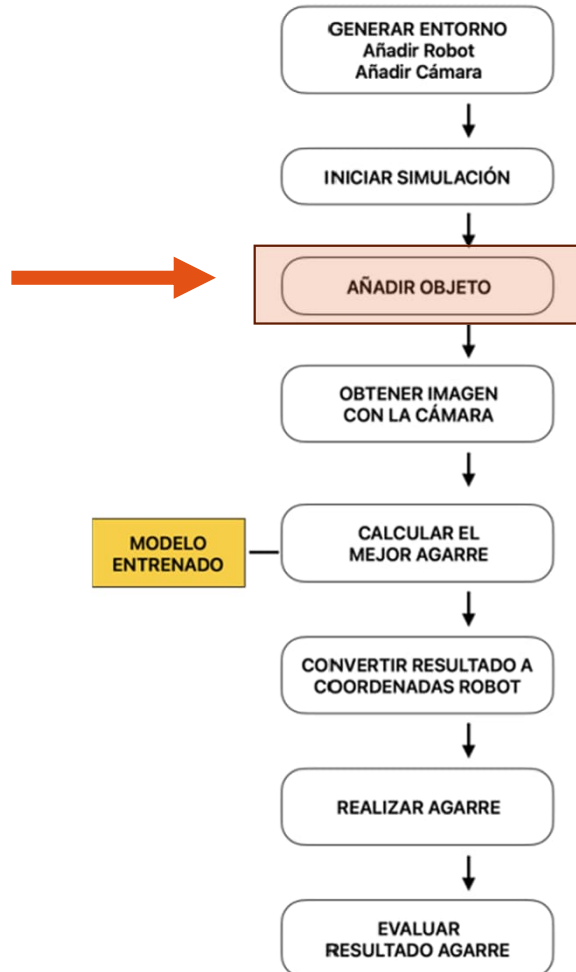


```
from isaacsim import SimulationApp
import omni

simulation_app = SimulationApp({"headless": False})
```



Simulación



```

x = random.uniform(0.38, 0.46)
y = random.uniform(0.40, 0.55)
pos_trabajo = [x, y, 0.08]

ang = random.randint(0, 359)
rotacion=[90, ang, 0]

path_modelos = "C:/TFM/modelos"

# Anadimos objeto a la escena
# pipon largo
# asset_path=path_modelos + "/IPAGearShaft2.usd"
# escala=[0.5, 0.5, 0.5]

# Biela
asset_path= path_modelos + "/121003320R--H.usd"
escala=[0.0015, 0.0015, 0.0015]

# casquillo
# asset_path= path_modelos + "/8201732034--B.usd"
# escala=[0.0025, 0.0025, 0.0025]

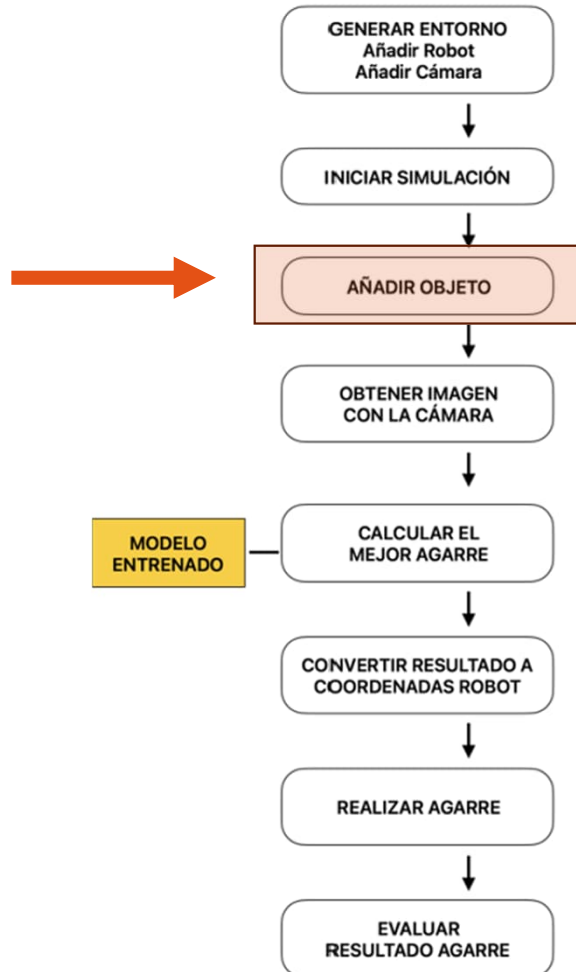
path_prim="/World/Objeto"
add_reference_to_stage(usd_path=asset_path, prim_path=path_prim)

grasp_obj = GeometryPrim(prim_paths_expr=path_prim,
                          scales=np.tile(np.array(escala), (1, 1)),
                          orientations=np.tile(np.array(rot_utils.euler_angles_to_quats(np.array(rotacion),
                          degrees=True, extrinsic = False)), (1, 1)),
                          positions=np.tile(np.array([pos_trabajo]), (1, 1)))

# Definimos objeto rigido para que adquiera fisicas
grasp_obj_rigid = RigidPrim(path_prim)
grasp_obj_rigid.enable_rigid_body_physics()
grasp_obj_rigid.set_masses(np.full(1,0.01)) # In kg

# Inicializamos el objeto en la posicion deseada
grasp_obj.initialize()
grasp_obj.set_world_poses()
  
```


Simulación



```

x = random.uniform(0.38, 0.46)
y = random.uniform(0.40, 0.55)
pos_trabajo = [x, y, 0.08]

ang = random.randint(0, 359)
rotacion=[90, ang, 0]

path_modelos = "C:/TFM/modelos"

# Anadimos objeto a la escena
# pipon largo
# asset_path=path_modelos + "/IPAGearShaft2.usd"
# escala=[0.5, 0.5, 0.5]

# Biela
asset_path= path_modelos + "/121003320R--H.usd"
escala=[0.0015, 0.0015, 0.0015]

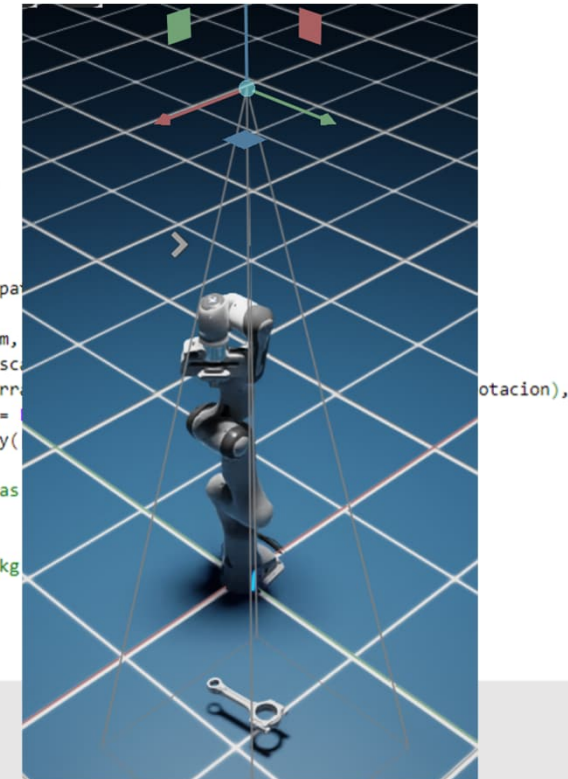
# casquillo
# asset_path= path_modelos + "/8201732034--B.usd"
# escala=[0.0025, 0.0025, 0.0025]

path_prim="/World/Objeto"
add_reference_to_stage(usd_path=asset_path, prim_path=path_prim)

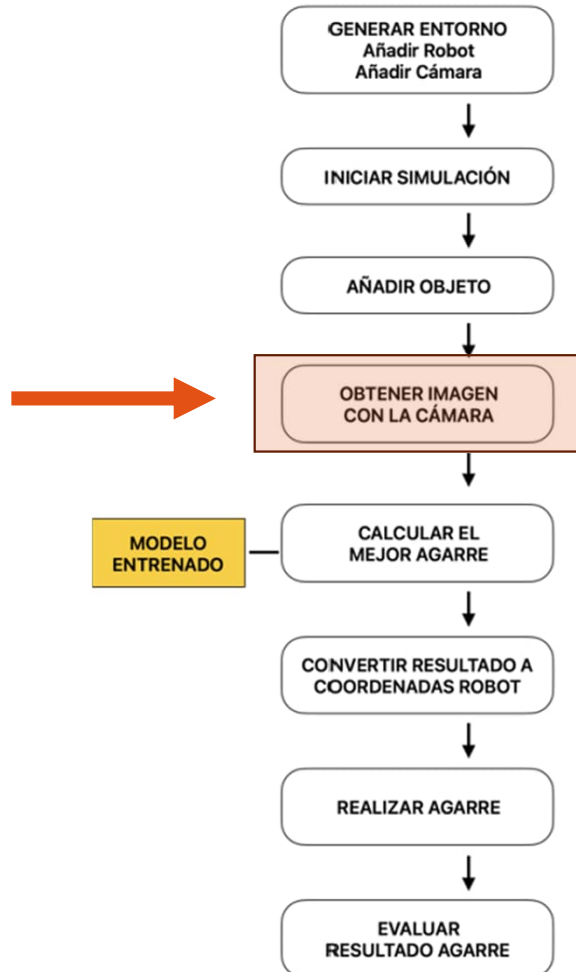
grasp_obj = GeometryPrim(prim_paths_expr=path_prim,
                          scales=np.tile(np.array(escala), 3),
                          orientations=np.tile(np.array(rotacion), 3),
                          degrees=True, extrinsic = False,
                          positions=np.tile(np.array(pos_trabajo), 3))

# Definimos objeto rigido para que adquiriera fisicas
grasp_obj_rigid = RigidPrim(path_prim)
grasp_obj_rigid.enable_rigid_body_physics()
grasp_obj_rigid.set_masses(np.full(1,0.01)) # In kg

# Inicializamos el objeto en la posicion deseada
grasp_obj.initialize()
grasp_obj.set_world_poses()
  
```



Simulación

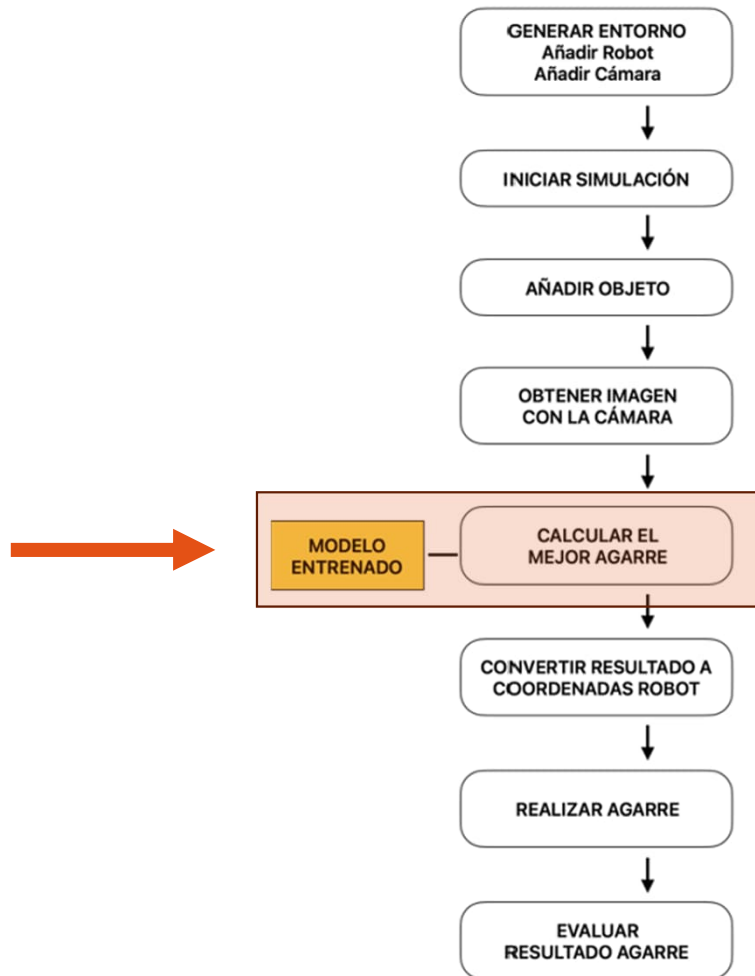


```

# Captura las imagenes con la cámara
ruta_imagenes = "C:/isaacsim/standalone_examples/api/isaacsim.robot.manipulators/imagenes/"
rgb_image = Image.fromarray(camera.get_rgb())
rgb_image.save(ruta_imagenes + 'imagen_rgb.png')
depth_image = camera.get_depth()
depth_aux = cv2.normalize(depth_image, None, 255, 0, cv2.NORM_MINMAX, dtype=cv2.CV_8U)
depth_normalized = Image.fromarray(depth_aux)
depth_colored = Image.fromarray(cv2.applyColorMap(depth_aux, cv2.COLORMAP_JET))
depth_normalized.save(ruta_imagenes + 'imagen_depth.tiff')
depth_colored.save(ruta_imagenes + 'imagen_depth_color.png')
  
```



Simulación



```
# Carga la red y calcula el mejor agarre
network = path_network
```

```
punto, angulo, ancho = calculo_agarre.mejor_agarre(network, n_test)
```

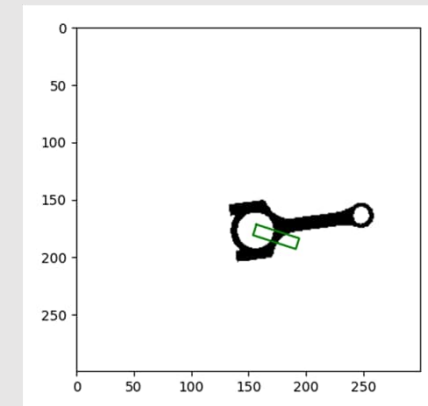
```
net = torch.load(network, weights_only = False)
device = torch.device("cuda:0")

depth_img = get_depth(os.path.join(ruta_imagenes + 'imagen_depth.tiff'))
x = numpy_to_torch(depth_img)

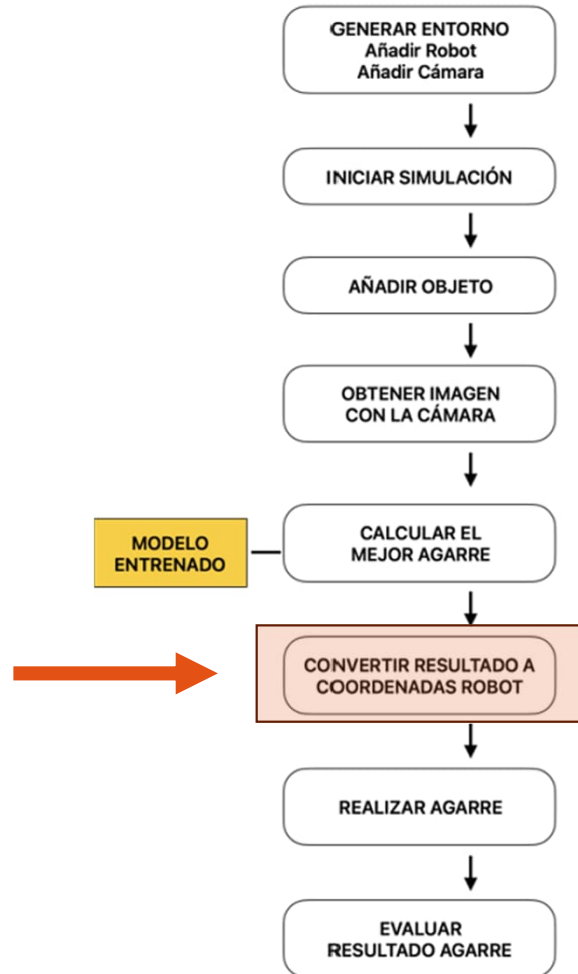
with torch.no_grad():
    xc = x.to(device)
    pred = net.predict(xc)
    q_img, ang_img, width_img = post_process_output(pred['pred']['pos'], pred['pred']['cos'],
                                                    pred['pred']['sin'], pred['pred']['width'])

    punto, angulo, ancho = plot_output_2(depth_img, q_img,
                                          ang_img, no_grasps=1, grasp_width_img=width_img)

return punto, angulo, ancho
```

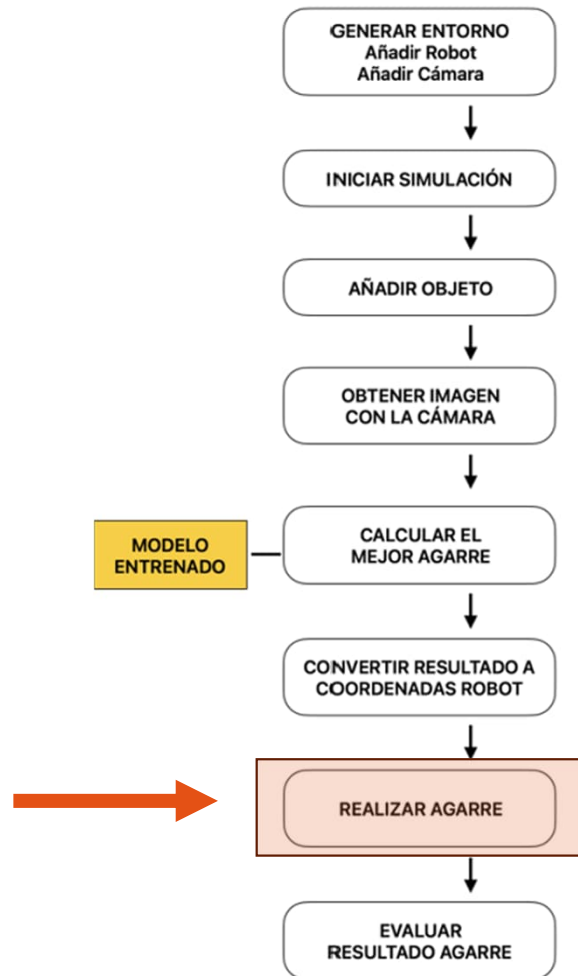


Simulación



```
picking = np.ndarray.flatten(grasp_obj.get_local_poses(indices=[0])[0])
picking[0] = 0.756 - punto[0] * 0.6 / 300
picking[1] = 0.845 - punto[1] * 0.6 / 300
picking[2] = 0.009
ang_picking = angulo * 180/np.pi
```

Simulación



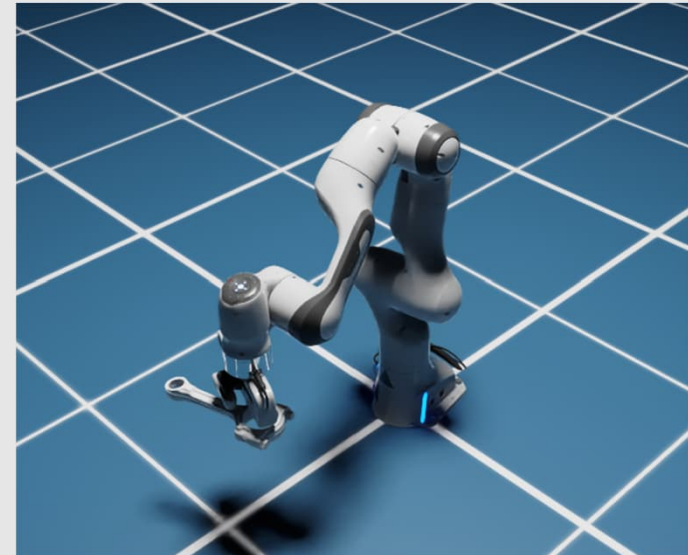
```

my_controller = PickPlaceController(
    name="pick_place_controller", gripper=my_franka.gripper, robot_articulation=my_franka
)

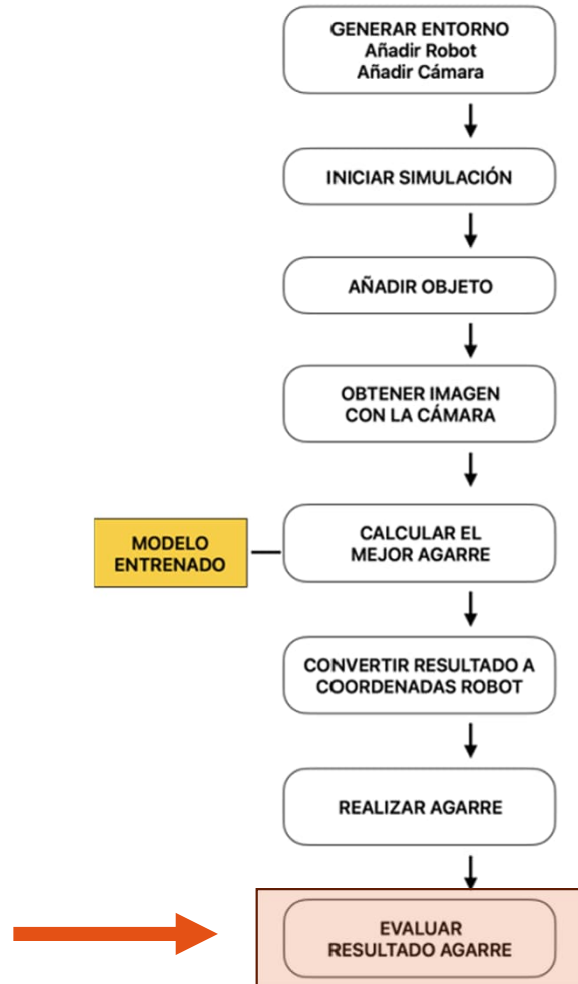
articulation_controller = my_franka.get_articulation_controller()

actions = my_controller.forward(
    picking_position=picking,
    placing_position = np.array([0.2, -0.5, 0.05]),
    current_joint_positions=my_franka.get_joint_positions(),
    end_effector_offset=np.array([0, 0.005, 0]),
    end_effector_orientation=rot_utils.euler_angles_to_quats(np.array([0, 180, ang_picking]), degrees=True),
)

```



Simulación

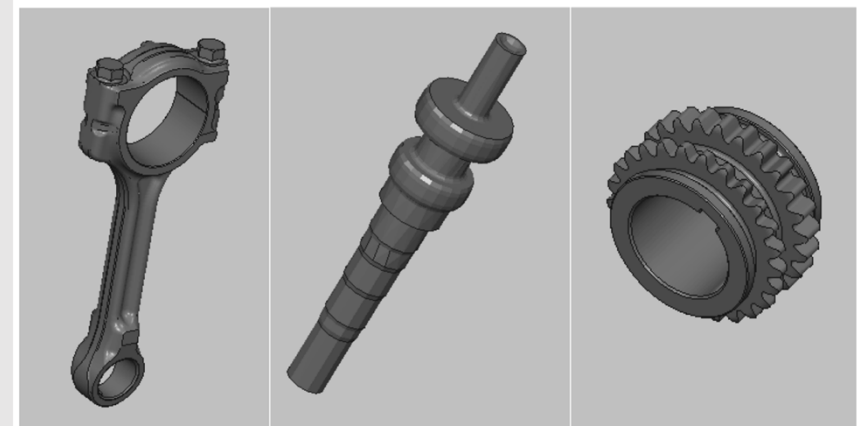


```

posicion = grasp_obj.get_local_poses()[0]
if (posicion[0,2]) > 0.1:
    agarre_parcial = True
if my_controller.is_done():
    print("done picking and placing")
    pos_actual = grasp_obj.get_local_poses()[0]
    a = [pos_actual[0,0], pos_actual[0,1]]
    b = [0.2, -0.5]

    agarre = False

    if np.allclose(a, b, atol=0.3):
        agarre = True
  
```



	Intentos	Completo	Parcial	No agarre
Objeto 1	100	76	8	16
Objeto 2	100	70	5	25
Objeto 3	100	65	10	25

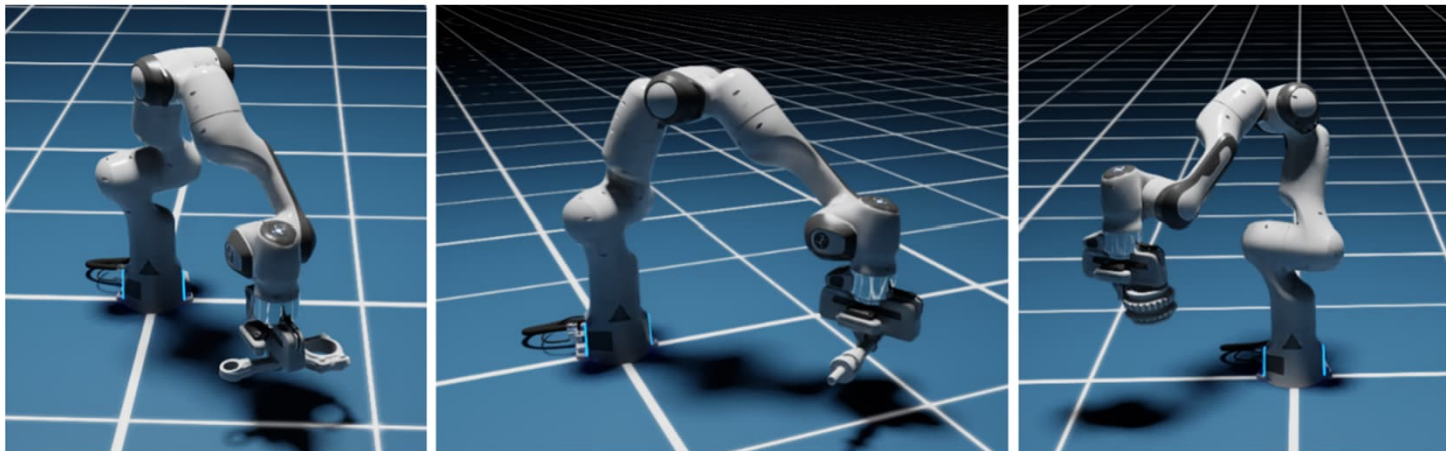
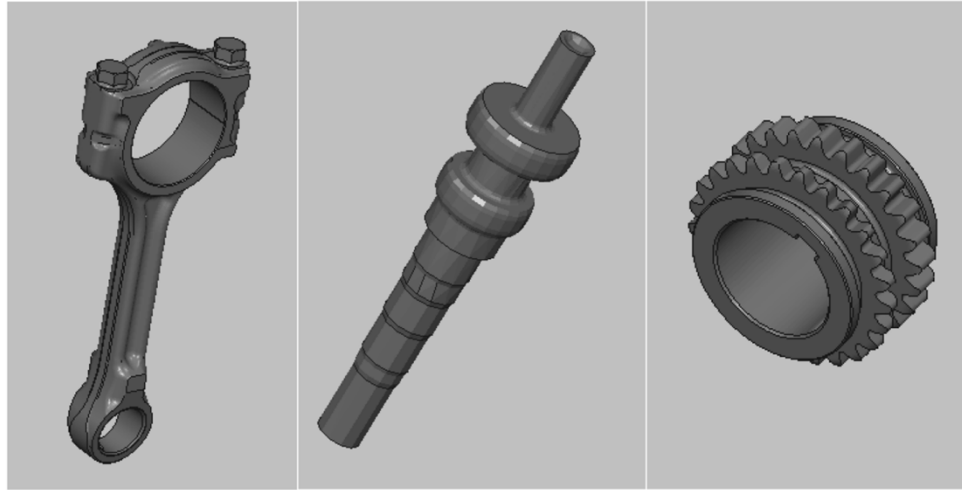
Conclusiones y Líneas Futuras

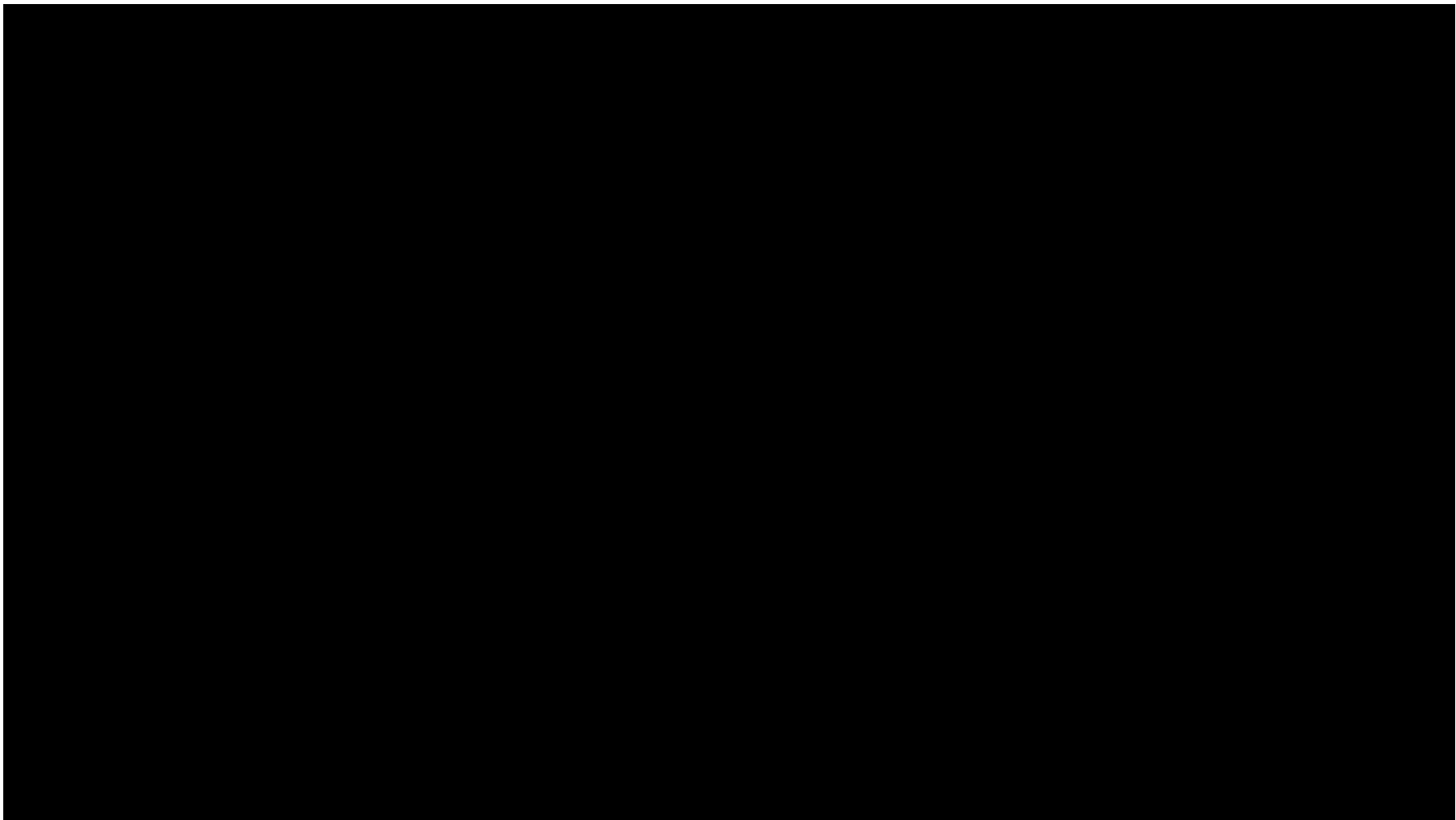
Conclusiones

- ✓ El agarre robótico es complicado en entornos no estructurados, en concreto en el sector industrial.
Las dificultades provienen de:
 - Variación en los objetos (tamaño, peso, material).
 - Presencia de otros elementos en el entorno.
 - Interacción humano-robot.
- ✓ Se desarrollaron y compararon varios modelos basados en redes neuronales:
 - Específicos para agarre robótico: GGCNN y GGCNN2.
 - Generales: ResNet y MobileNet.
 - GGCNN2 fue el que ofreció mejores resultados según las métricas de evaluación.
- ✓ La simulación permite probar múltiples escenarios sin necesidad de un sistema físico:
 - Requiere un equipo con alta capacidad computacional.
 - A largo plazo, ahorra tiempo, recursos y dinero.
- ✓ El uso combinado de datasets, algoritmos de IA y simuladores permite realizar:
 - Entrenamientos realistas.
 - Evaluaciones eficientes de estrategias de agarre en entornos variados y no estructurados.

Líneas Futuras

- Exploración de nuevos simuladores menor coste computacional como: PyBullet, Mujoco, Webots.
- Simulaciones más realistas. Incorporar elementos industriales reales en las simulaciones como:
 - Cintas transportadoras.
 - Vehículos autoguiados (AGVs).
 - Sensores de alta precisión.
- Simulaciones de agarre de objetos blandos incluyendo un control de fuerza en la garra del robot
- Integrar algoritmos en sistemas reales con PLC (Controladores Lógicos Programables).
- Explorar algoritmos de aprendizaje por refuerzo como:
 - PPO (Proximal Policy Optimization)
 - DDPG (Deep Deterministic Policy Gradient)





Muchas gracias