Отчёт по домашней работе №2 «Task Space Inverse Dynamics»

1. Уравнения, описывающие TSID контроллер

$\dot{p} = J(q) \cdot \dot{q}$

$\ddot{p} = J(q) \cdot \ddot{q} + \dot{J}(q) \cdot \dot{q}$

$a_q = J^{-1}(q)\left\{\begin{bmatrix} a_x \\ a_\omega \end{bmatrix} - \dot{J}(q) \cdot \dot{q}\right\}$, где:

$\ddot{r} = a_r$

$\dot{\omega} = a_\omega$

$\dot{R} = S(\omega)R$

Логарифмическая карта ошибки ориентации:

$R_{err} = R_d R^T$

$S_{err} = log\ (R_{err})$

Управляющий момент τ:

$\tau = M \cdot a_q + F_{nle}$

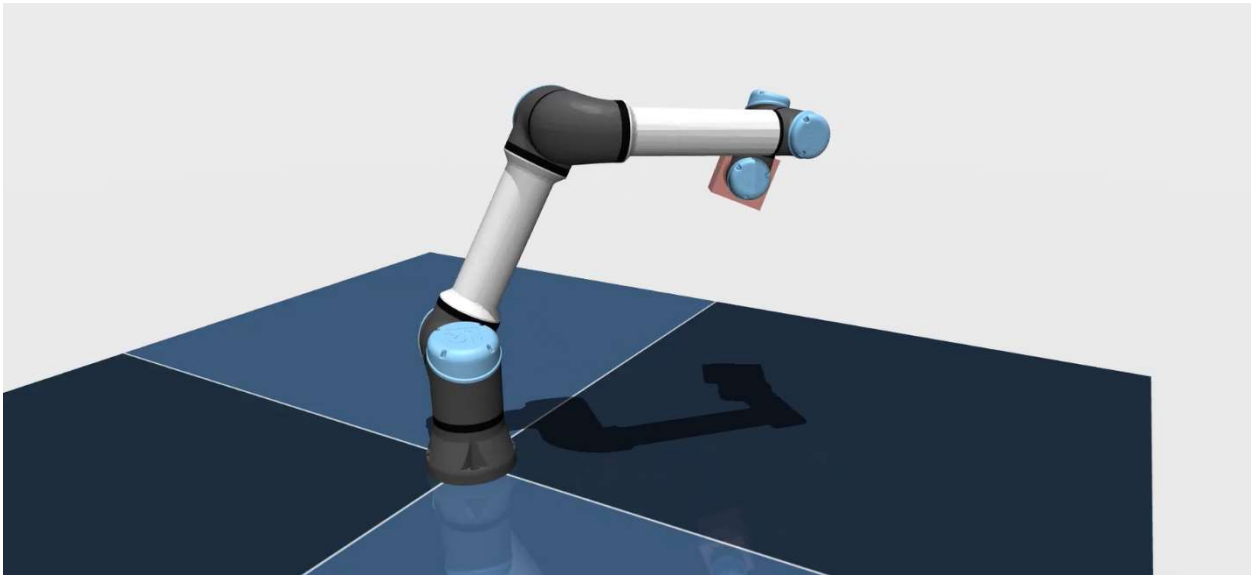
Реализация управления в python:

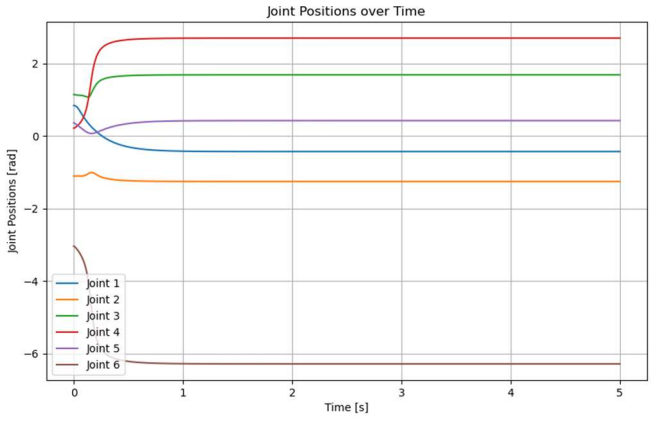
```
def tsid_controller(q: np.ndarray, dq: np.ndarray, t: float, desired: Dict) ->
np.ndarray:
    """Task-space controller for the robot."""

    pin.computeAllTerms(model, data, q, dq)
    M = data.M
    nle = data.nle

    kp = np.array([1000, 1000, 1000])
    kd = np.array([200, 200, 200])

    # Desired pose
    desired_position = desired['pos']
    desired_quaternion = desired['quat']

    # Compute current end-effector pose using Pinocchio
    pin.forwardKinematics(model, data, q, dq)
    ee_frame_id = model.getFrameId("end_effector")
    frame = pin.LOCAL
    pin.updateFramePlacement(model, data, ee_frame_id)

    # Get velocities
    twist = pin.getFrameVelocity(model, data, ee_frame_id, frame)

    # Jacobian
    J = pin.getFrameJacobian(model, data, ee_frame_id, frame)
    dJ = pin.getFrameJacobianTimeVariation(model, data, ee_frame_id, frame)

    ee_pose = data.oMf[ee_frame_id]
    ee_position = ee_pose.translation
    ee_rotation = ee_pose.rotation

    # Compute position and orientation errors
    position_error = desired_position - ee_position

    # Convert quaternion to rotation matrix and compute orientation error
    desired_rotation_matrix = quaternion_to_rotation_matrix(desired_quaternion)
```

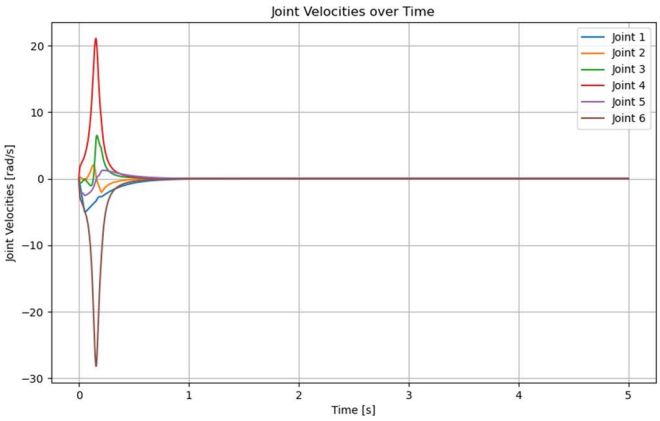
```
    # Corrected orientation error calculation
    orientation_error = pin.log(desired_rotation_matrix @ ee_rotation.T)
    pose_err = np.zeros(6)
    pose_err[:3] = position_error
    pose_err[3:] = orientation_error

    # Calculate desired accelerations using the outer loop control equation
    a_x = kp * position_error + kd * (0 - twist.linear)
    a_w = kp * orientation_error + kd * (0 - twist.angular)

    desired_acceleration = np.concatenate((a_x, a_w))

    if np.linalg.det(J) == 0:
        J_inv = np.linalg.pinv(J)
    else:
        J_inv = np.linalg.inv(J)

    ddq = J_inv @ (desired_acceleration - dJ @ dq)

    tau = nle + M @ ddq

    return tau
```

2. Вывод графиков положений и скоростей

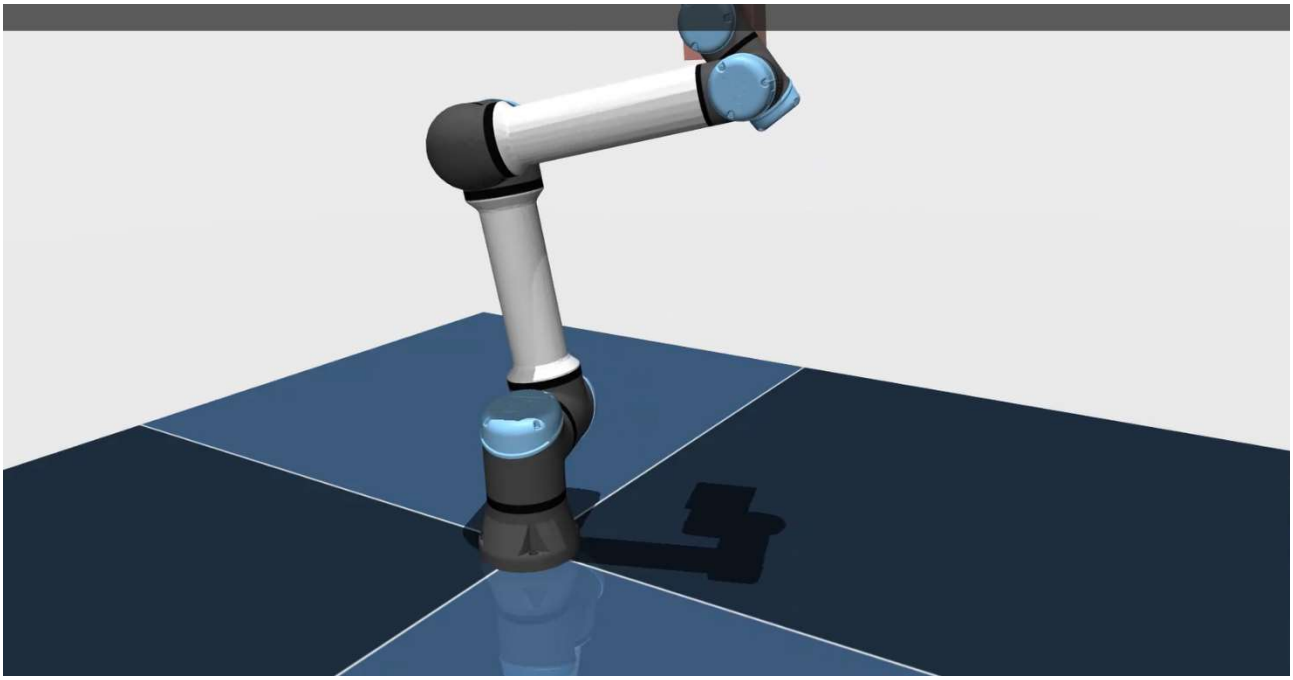
Перемещение энд-эффектора в 1 положение (05_tsid1.mp4):

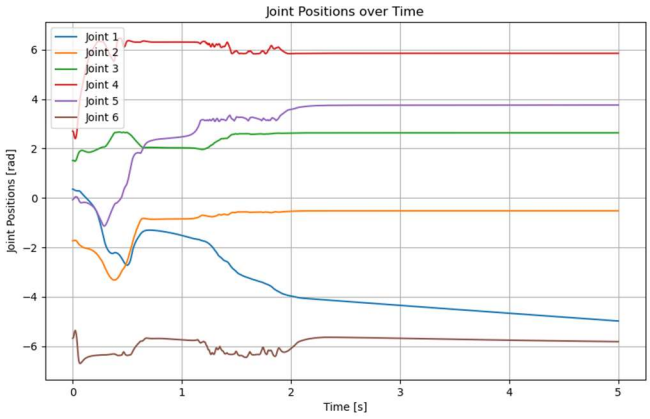


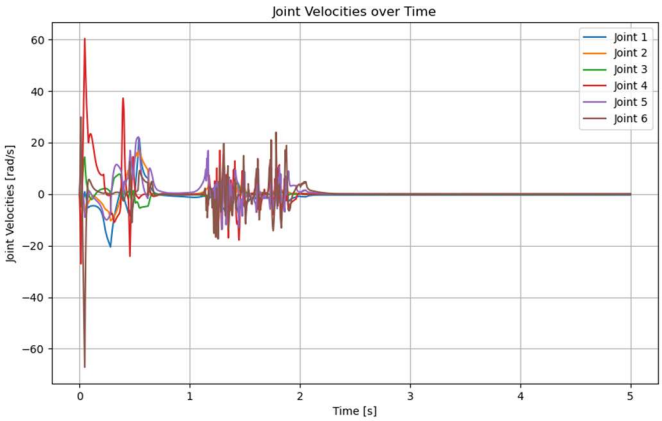




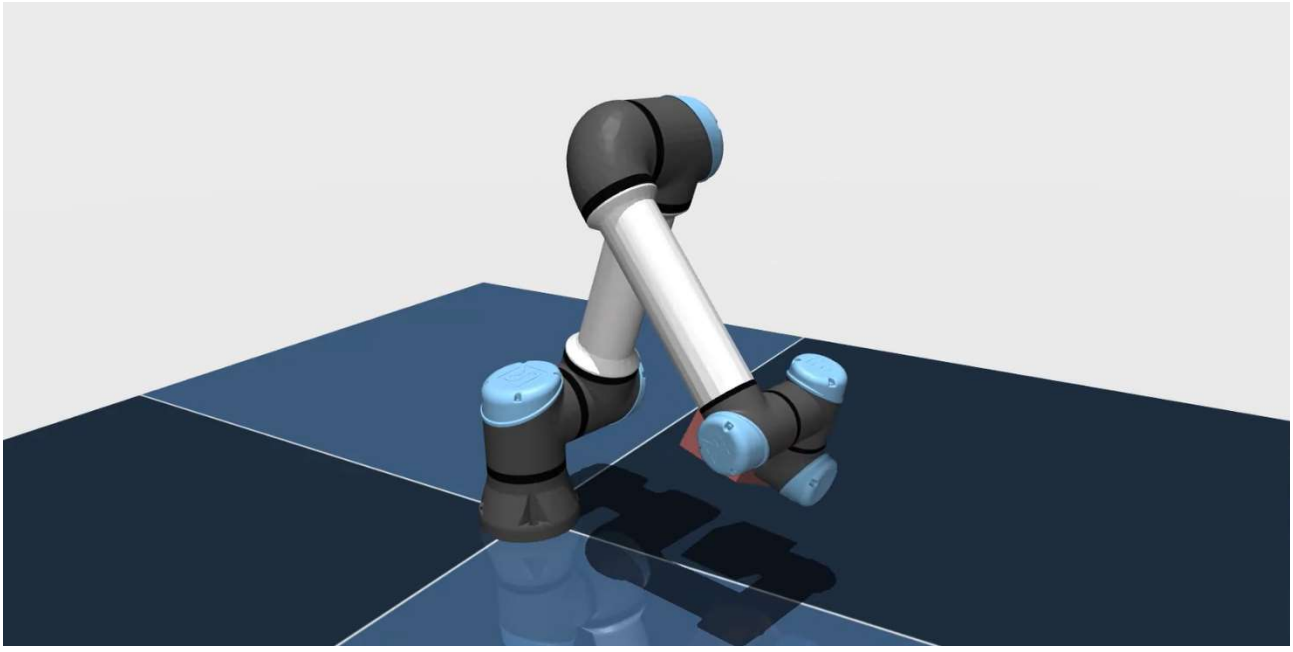
Перемещение энд-эффектора во 2 положение (05_tsid2.mp4):

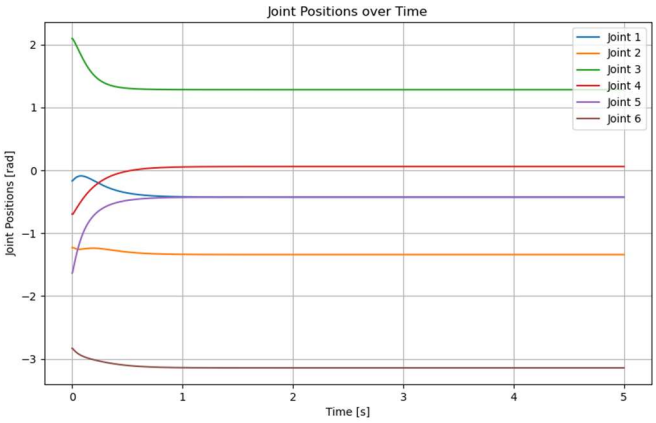


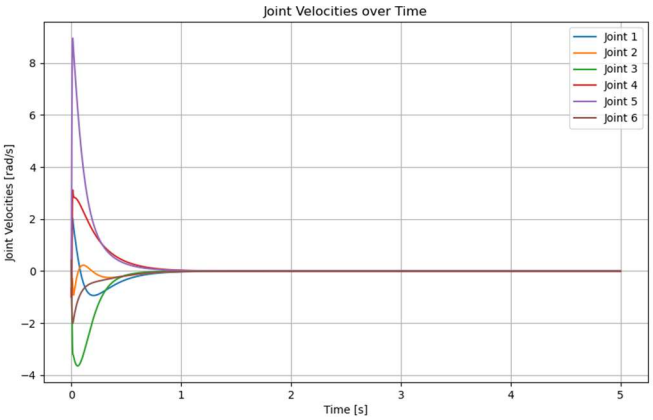




Перемещение энд-эффектора в 3 положение (05_tsid3.mp4):







3. Движение по круговой траектории представлено в файле 05_circ_tsid.mp4