Robotics Sensing and Navigation Final Report: Group 3

Fall 2024

Oliver Hugh, Jack Gladowsky, Zhang Ye, Sasha Oswald, Shail Jagat Mehta

Introduction

To learn more about modern Simultaneous Localization and Mapping (SLAM) algorithms, our group looked at LIO-SAM, which is tightly-coupled LiDAR Inertial Odometry via Smoothing and Mapping [1]. We researched this algorithm, examined its corresponding GitHub repository [2], implemented the algorithm on provided datasets, and then collected our own data to test the algorithm out on. Data was collected with Professor Hanumant Singh using Northeastern's autonomous car, NUANCE, to drive around the local area. After collecting this data, our group decided to also look at FAST-LIO [3], an alternative to LIO-SAM that has a higher emphasis on computational efficiency and robustness in use in fast-moving systems in noisy and cluttered environments.

Research and Background

We began by reading [1] and learning about LIO-SAM. It is a highly accurate SLAM algorithm that couples LiDAR scans with IMU data to estimate the trajectory of a mobile robot or system and to perform map building. LIO-SAM employs 4 different factors to estimate the robot's state and to create a factor graph consisting of the robot's state representation, as well as those factors. Its state is represented by a rotation matrix, and 3-dimensional vectors of its position, velocity, and sensor bias. Figure 1 below shows the diagram provided in [1] that represents how LIO-SAM works.

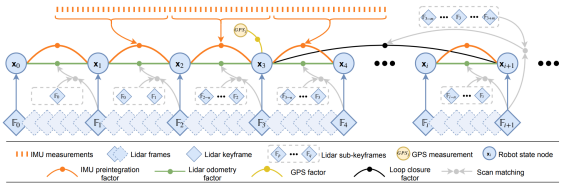

Figure 1: LIO-SAM Algorithm Diagram from Paper

The four factors include an IMU pre-integration factor, a LiDAR odometry factor, a GPS factor, and a loop closure factor. The IMU factor is very simple to understand, as it is simple kinematic equations that use the measurements of the IMU to calculate the robot's position and velocity. Conceptually, it simply uses kinematic equations, but instead of using solely raw measurements, it combines these raw measurements with an estimate of the noise present in the system, as well as the moving estimate of the IMU's bias.

The LiDAR factor is very interesting, as the algorithm uses the sliding keyframes method to reduce its computational load. The algorithm takes the LiDAR scan and extracts edge and planar features by looking at the roughness of points in localized regions, as, intuitively, edges will have a comparatively large roughness, whereas planes/planar features will be characterized by a small roughness. It puts these features into a LiDAR frame. The sliding keyframe method is described in detail in [1], and the general principle is that between successive LiDAR scans, there is not much difference in the robot's position or pose, meaning not every single scan should be used for calculations. So, a threshold is set, and once the robot's position or pose have changed more than that defined threshold from the previous keyframe, the LiDAR frame is added as a new keyframe to the factor graph and incorporated into new measurements. For [1], the threshold is

set as a change in position of 1 meter or more, and a change in pose of 10 degrees or more. Once a new LiDAR frame is added as a node to the factor graph, the algorithm performs scan matching and converts the LiDAR frame from the body of the robot's frame to the world frame and matches it to the algorithm's global voxel map. It then finds the optimal relative transformation between this keyframe and the last to incorporate into the state estimation of the robot, as well as the mapping of its environment.

The GPS factor is very helpful when available, but not necessary for the algorithm and completely optional. Additionally, the name of 'GPS factor' is a bit misleading, as [1] states that the algorithm can actually take a variety of absolute measurements, such as from an altimeter or a compass, opposed to only a GPS. While the algorithm has pretty low drift, it does still drift in the long term on longer routes, so having the GPS factor is helpful for correcting this long term drift. For efficiency, the algorithm does not take every GPS measurement into account when doing calculations, but instead only when the estimated position covariance without the absolute measurement is greater than the covariance of the GPS or absolute measurement point.

The final factor, the closure factor, is another important part of this algorithm and a big reason why it is so accurate compared to some of its LiDAR-based predecessors. As the robot moves and adds nodes to its factor graph, the algorithm goes through the graph and determines if any past states are near the current frame in Euclidean space. If so, LIO-SAM then tries to perform scan matching to correct both mapping and trajectory estimates. While [2]'s implementation of LIO-SAM's loop closure is Euclidean based, [1] discusses how there are other methods that could work, such as using place recognition with 3D point clouds or by using the intensity of the LiDAR measurements.

Compared to past LiDAR-based SLAM algorithms, LIO-SAM is highly accurate and is capable of real-time performance. Despite some of the strategies discussed that help reduce its computational load, there are algorithms out there with better real-time performance, such as FAST-LIO.

After some issues with getting LIO-SAM working for the collected dataset, which will be explained later on in this report, our group decided to also look at FAST-LIO by reading [3] and looking at the corresponding GitHub repository at [4]. Just like LIO-SAM, FAST-LIO is a tightly coupled LiDAR Inertial odometry algorithm, which means it uses IMU and LiDAR measurements together rather than completely processing them separately. FAST-LIO was created with an emphasis on real-time performance and computational efficiency, and is perfect for scenarios in which the system is moving rapidly in a noisy environment. The diagram provided in [3] that shows its general methodology is presented below in Figure 2.

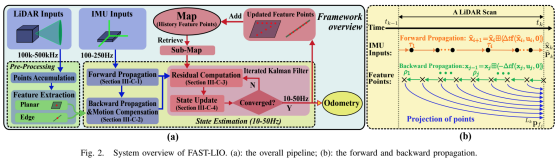

Fig. 2. System overview of FAST-LIO. (a): the overall pipeline; (b): the forward and backward propagation.

Figure 2: FAST-LIO Algorithm Diagram from Paper

This algorithm is a great algorithm for high speed drones or unmanned aerial vehicles (UAVs). While LIO-SAM can be executed in real time, it may be too slow if it is in a feature dense environment, if the system is moving quickly, or if the system does not have a high enough computational load. These last two

cases are very common when it comes to using UAVs, which is why its efficiency as well as its ability to deal with fast motion and noisy and cluttered environments is so powerful.

Unlike LIO-SAM, FAST-LIO does not use a factor graph, which makes it faster, but also decreases its modularity, as it is unable to take in absolute measurements or easily use different methods for the different factor graph integration steps described previously in this report. It extracts features from its LiDAR, but unlike in LIO-SAM which processes it in batches by looking at the whole scan at once, FAST-LIO instead begins its computations incrementally as LiDAR points come in.

It propagates IMU measurements forward to estimate the position and pose of the system, as well as to compensate for the motion during a LiDAR scan and propagates LiDAR features backwards to refine state estimates and bias from the IMU. FAST-LIO uses an iterated extended Kalman filter, and to achieve such high computational efficiency, [3] derives and presents a new equivalent formula for the Kalman gain. In their equivalent formula, instead of inverting a large matrix with dimensions equal to the number of measurements, they instead only have to invert a matrix with dimensions equal to the state of the system. This, combined with incremental calculations of the Kalman gain help FAST-LIO achieve its high efficiency.

Between these two algorithms, one major deciding factor in which one would be best to use in a specific system would be a question of one major trade-off - whether accuracy or efficiency is more important. LIO-SAM is highly accurate and has relatively low long term drift - especially if a GPS or some other type of absolute measurement is available for correction. That being said, if the system is expected to move fast, have low computational power, or its environment is expected to be especially noisy or feature dense, then FAST-LIO is a great choice. While it is unable to incorporate those absolute measurements, and does not use loop closure to correct measurements, it is still a very accurate SLAM algorithm, which will be explored more in depth later in this report.

Running Algorithms on Example Datasets

To get started with LIO-SAM, we first cloned the repository onto our own machines. We then used the included Dockerfile to get a container built that is configured to use ROS1 Kinetic. The container also clones and builds the LIO-SAM repository, so everything is ready to use out-of-the-box. Once the LIO-SAM repository was set-up, we downloaded a few test datasets such as the example datasets of a park and the MIT campus, and also a dataset that was taken from ISEC at Northeastern University. These datasets have a good variety as they show outdoor nature environments, outdoor urban environments, and also large, complex indoor spaces.

We first decided to run the MIT campus dataset as this was included in the repository and would work without changes to the parameters. This environment is a great test as a common usage will be in busy city settings as shown in the images in figure 3.

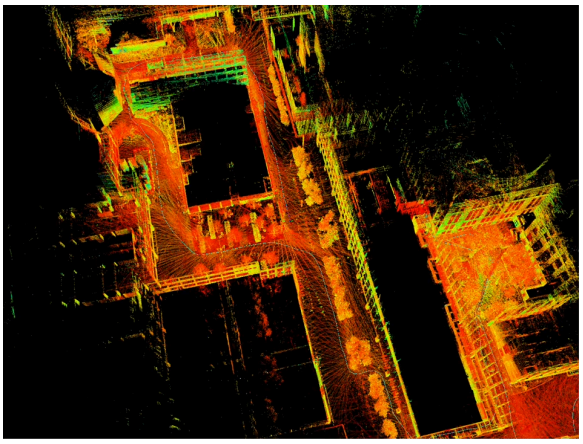

Figure 3. MIT Campus LIO-SAM Map

Next, we ran the same dataset again, but this time we enabled the loop closure feature that LIO-SAM implements using ICP. Figure 4 shows the results on the same dataset from the LIO-SAM paper, and it shows how the loop closure of LIO-SAM works compared to other SLAM algorithms. We can see in figure 5 how the robot returns to its starting point where the trajectory loop is closed.

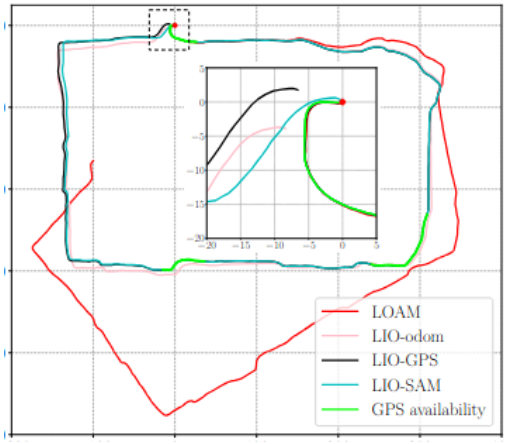

Figure 3. MIT Campus Trajectory Comparison

Finally, we used an open source dataset of the ISEC building at Northeastern University to test how LIO-SAM performs on complex, large, indoor spaces with lots of windows and reflections. To get this dataset working, we had to edit the `params.yaml` file to configure LIO-SAM to recognize the LIDAR and IMU sensors and topics that are used to collect the dataset. We can see in figure 6 that the generated map still has fairly high accuracy, but is a lot more susceptible to ghosting artifacts.

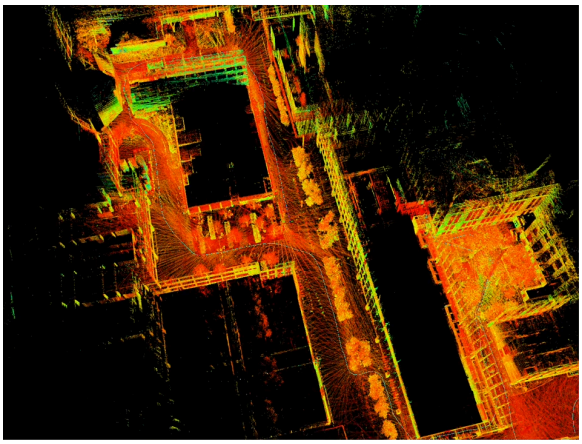

Figure 5. MIT Campus LIO-SAM Loop Closure

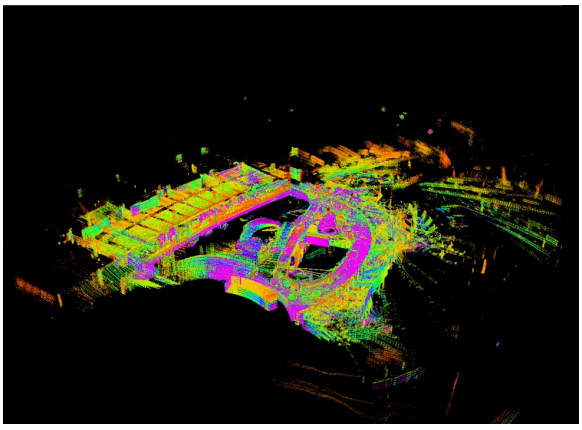

Figure 6. ISEC First Floor LIO-SAM Map

We then tried to implement FASTLIO, and to do this, we created a script to build a Docker container suitable for running FASTLIO. The `docker run` command uses an image from Docker Hub, ensuring that the container's Ubuntu version is preferably 18.04 and that the ROS version is ROS1 Melodic or higher. Importantly, before running the launch file from the source code, it is necessary to install `livox_ros_driver`. Next, the script automatically creates a workspace, clones the repository, and compiles it using `catkin_make`. For our dataset, we primarily used indoor data collected from the first floor of

the ISEC building at Northeastern University, driving data from Boston streets collected through NUance, and driving data from the Renaissance parking garage.

Once our workspace is successfully compiled, we first use the `rosbag info` command to examine the topics in the rosbags to identify the required point cloud and IMU data topics. Then, by editing the `ouster64.yaml` file in the FASTLIO repository (since all our datasets were collected using Ouster sensors), we modified the default point cloud and IMU topics and enabled the path visualization. After sourcing the environment, we can launch the launch file and see RViz appear on the screen. Next, we open a new terminal, enter the Docker container, and play back the dataset.

While running FASTLIO on our own datasets, we discovered an issue when using the Vectornav IMU 9-axis sensor: the program encountered IMU and LiDAR time synchronization problems. This led to poor results in RViz, where the point clouds appeared chaotic, and the trajectory was drifting and not smooth. As shown in the figure 7, the point cloud is highly disorganized, and the trajectory exhibits significant drift and instability. This highlights that time synchronization is critical for FASTLIO. Subsequently, we switched the IMU topic to the Ouster built-in IMU data (Ouster's built-in IMU is a 6-axis sensor and has minimal delay relative to the Ouster LiDAR). As a result, we obtained a very stable and consistent trajectory, as well as point clouds with fine details, as shown in the figure 8.

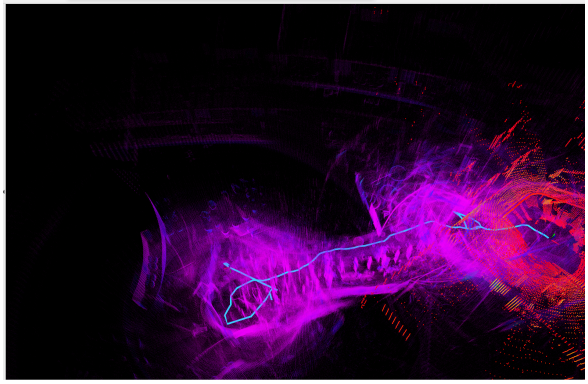

Figure 7: FAST-LIO with ISEC_data (9 DOF IMU)

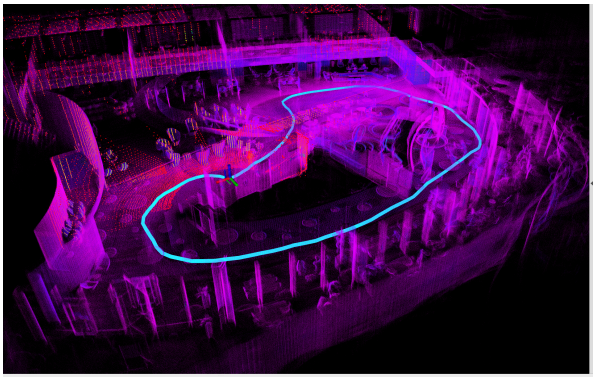

Figure 8: FAST-LIO with ISEC_data(6 DOF IMU)

Collected Data and Analysis

On Tuesday, November 26, our group met Professor Singh at EXP to collect data with Northeastern's autonomous car, NUANCE. There were two groups present, and between us, three datasets were collected. We collected the first dataset, which involved driving around nearby streets in the local neighborhood. This dataset involved us leaving EXP and driving down Columbus Ave. From there, we drove past Columbus Ave.'s intersection with Massachusetts Ave. and took a right to drive in a square twice to provide an extra loop in our trajectory. After driving around the same street twice for added loop closure when we ran the algorithm, we then headed back to where we came to close the loop of our overall path.

The second group then collected a similar dataset with Professor Singh, and when they

came back, we wanted to collect one final dataset. This time, instead of driving around a Boston neighborhood, we wanted to drive through Renaissance garage. This parking garage is about 10 stories tall, and we wanted to collect a dataset in which the car would be moving where its altitude would significantly change and to see how the algorithm would perform on a truly three dimensional dataset.

Our initial project was simply to analyze LIO-SAM; however, even though NUANCE has a large suite of sensors, there were some issues with its setup that prevented us from running LIO-SAM on the collected datasets. The sensors in the car that we were primarily interested in were the LiDAR and IMU. NUANCE has two separate IMU's - a 6 degree of freedom (DOF) and a 9 DOF sensor. In the recorded ROSBAG, the LiDAR and the 6DOF IMU are on a common clock. Unfortunately, the 9DOF IMU published data with a separate clock. With no way to accurately relate these two clocks, we are unfortunately unable to use the data from the 9DOF IMU with the LiDAR data. Since the LIO-SAM algorithm does not currently support 6-axis IMUs (such as the Ouster built-in IMU), it meant that we would be unable to run it on the NUANCE data.

To process these valuable datasets, we turned our attention to FASTLIO, which supports 6-DOF IMUs, and eventually obtained excellent output results.

Our collected data allowed us the chance to comprehensively evaluate the performance of FASTLIO, as the NUANCE provided a large dataset with sufficient duration: an 11-minute dataset of Boston street driving and a dataset of driving through a large parking garage.

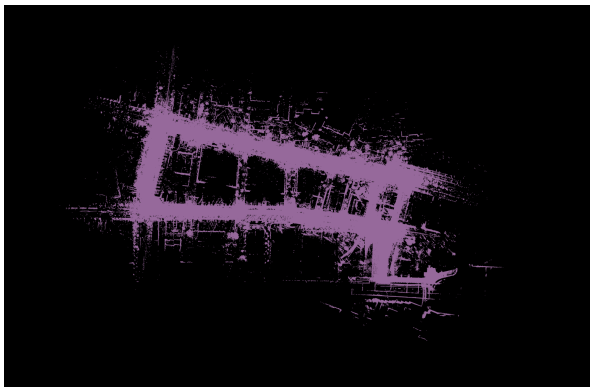

Figure 9: FAST-LIO with NUance_Street_Data

The point cloud map generated by FASTLIO from the Boston driving dataset is shown in Figure 9, and the street driving trajectory is shown in Figure 10. The overall clarity of the mapping point cloud is relatively good, with complete building outlines, continuous road structures, and well-preserved local features (such as utility poles and sidewalks). However, we observed a noticeable drift phenomenon. The most significant issue is the lack of proper loop closure—i.e., the endpoint does not align with the starting point. This is caused by cumulative odometry errors, with an estimated drift of approximately 10–15 meters.

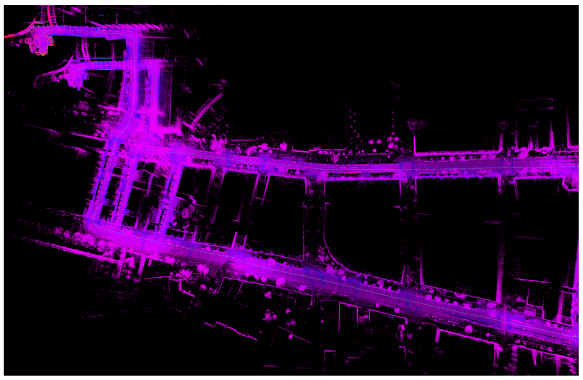

Figure 10: NUance street driving trajectory with FASTLIO

Additionally, by examining the side-view PCD scan map (as shown in Figure 11), we noticed that the entire map does not lie on a single plane. This indirectly indicates that the

cumulative error in the Z-axis of the IMU caused the elevation of the map to continually rise, resulting in a "double-layered" map. These issues are primarily caused by the lack of loop closure detection and optimization, as well as the accumulation of IMU integration errors.

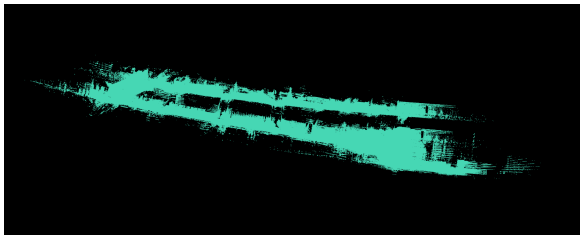

Figure 11: Side view of NUance trajectory with FASTLIO

The same issue occurred with the parking garage dataset. As is somewhat visible in figures 12 and 13, during the vehicle's ascent on the ramp, the surrounding garage structures are clearly visible, and the height of each floor is nearly consistent. However, during the vehicle's descent on the return path, the trajectory exhibits vertical deviations, and the point cloud data shows ghosting artifacts. This is, once again, caused by the accumulation of IMU odometry errors and the absence of a loop closure detection module. On a technical level, we could consider adjusting the feature extraction and matching parameters or even adding a loop closure detection module to address these problems.

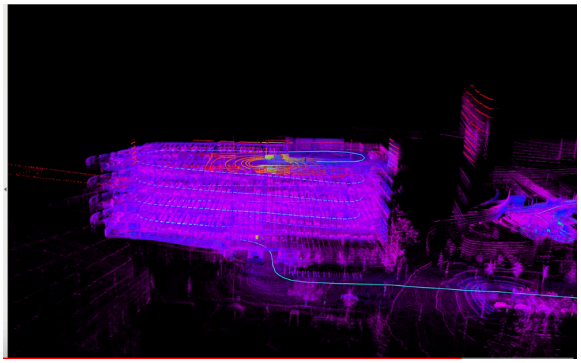

Figure 12: FAST-LIO with NUance_Garage_Data Screen Capture 1

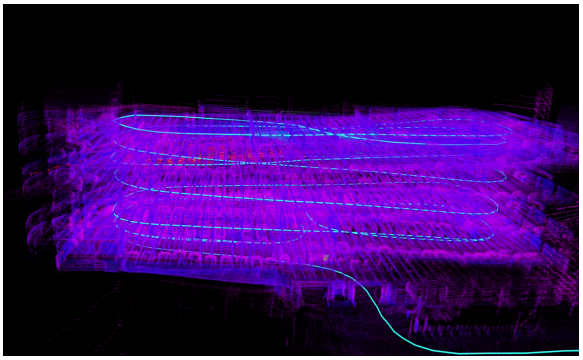

Figure 13: NUance driving garage trajectory with FASTLIO Screen Capture 2

Conclusion

As is the case with all fields of engineering, choosing between different SLAM algorithms often comes down to a question of trade-offs. LIO-SAM and FAST-LIO are both great algorithms that are very accurate, but they are not a universal solution to every system or SLAM problem. FAST-LIO is a highly efficient algorithm that can be performed in real-time when computing power is comparatively limited, works well in noisy environments, and can handle motion distortion from a fast moving system. It provides a good estimate of trajectory and creates a pretty accurate map, but it does not employ loop closure and has a drift problem. It is efficient but it is not modular, unlike LIO-SAM which employs factor graphs, can take in different absolute measurements, can utilize different methods of fusing LiDAR and IMU data, and can use loop closure for more accurate trajectory estimates and maps. However, LIO-SAM requires more computing power and time, but it could be used in real-time depending on the system and its requirements. Ultimately, neither one is objectively better than the other as a whole, and they excel at different things. System requirements and design objectives should drive the decision as to whether to use LIO-SAM, FAST-LIO, or perhaps even some other SLAM algorithm. This project allowed us to learn more about modern

SLAM algorithms by researching FAST-LIO and LIO-SAM, running sample datasets, and then collecting our own data to verify and compare output results ourselves.

References

[1] T. Shan, B. Englot, D. Meyers, W. Wang, C. Ratti, and D. Rus, “LIO-SAM: Tightly-coupled Lidar Inertial Odometry via Smoothing and Mapping,” 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Oct. 2020, doi: https://doi.org/10.1109/iros45743.2020.9341176.

[2] T. Shan, “LIO-SAM,” GitHub, Nov. 08, 2022. https://github.com/TixiaoShan/LIO-SAM

[3] W. Xu and F. Zhang, “FAST-LIO: A Fast, Robust LiDAR-inertial Odometry Package by Tightly-Coupled Iterated Kalman Filter,” IEEE Robotics and Automation Letters, pp. 1–1, 2021, doi: https://doi.org/10.1109/lra.2021.3064227.

[4] “hku-mars/FAST_LIO,” GitHub, Jan. 30, 2023. https://github.com/hku-mars/FAST_LIO