

Rapport d'un projet IoT

Intitulé :

***Système de suivi énergétique
solaire PV basé sur l' IoT***

Département : Informatique

Matière : Architecture des ordinateurs

Niveau : 1ère année génie informatique

Réalisé par : Wassim OMRANE - Ismail BOUJELBEN - Wafa KATAR

Enseignant : Tarak FRIKHA

Année Universitaire : 2025 - 2026

Résumé

Ce rapport présente la conception, la réalisation et l'analyse d'un système de suivi énergétique solaire photovoltaïque (PV) basé sur l'Internet des Objets (IoT). Le projet utilise un panneau solaire de 6V 3W comme source d'énergie, un microcontrôleur ESP32 pour l'acquisition et le traitement des données, et la plateforme ThingSpeak pour la visualisation en temps réel. Les paramètres mesurés incluent la tension, le courant (via ACS712), la puissance calculée, la température (LM35) et la luminosité (LDR). L'énergie est stockée dans une batterie Li-ion 18650 gérée par un chargeur MPPT (CN3065), et le système est alimenté de manière autonome via un boost converter. Une simulation de charge est effectuée avec des LEDs configurées comme un feu tricolore. Les objectifs sont éducatifs et démonstratifs, visant à illustrer l'efficacité des systèmes solaires connectés. Les tests montrent une puissance maximale d'environ 2,8W en plein soleil, avec une précision des mesures améliorée par un moyennage des lectures ADC. Malgré des limitations comme le bruit sur l'ADC de l'ESP32, le projet démontre le potentiel de l'IoT pour le monitoring énergétique renouvelable, ouvrant des perspectives pour des applications plus avancées.

(Mots-clés : Énergie solaire, IoT, ESP32, ThingSpeak, Monitoring énergétique)

Table des matières

1	Conception et Spécification	3
	Introduction	4
1.1	Analyse des Besoins	4
1.2	Schéma Fonctionnel	5
1.3	Outils et environnement de développement	6
	Arduino IDE	6
	ThingSpeak	7
	Langage de programmation : C++	7
	Conclusion	8
2	Réalisation et Mise en Œuvre	9
	Introduction	10
2.1	Montage Physique	10
2.2	Programmation et Configuration	13
	2.2.1 Structure et principes du code	13
	2.2.2 Algorithmes clés implémentés	13
	Remarque :	14
	2.2.3 Configuration ThingSpeak	14
2.3	Tests et Calibration	15
	Conclusion	16
3	Résultats, Analyse et Perspectives	17
	Introduction	18
3.1	Résultats Obtenus	18
3.2	Limitations et Améliorations	18

3.3 Perspectives	19
Conclusion	20

Table des figures

1.1	Schéma fonctionnel global du système	6
1.2	Logo officiel de l'Arduino IDE	7
1.3	Logo officiel de ThingSpeak	7
1.4	Logo officiel du langage C++	8
2.1	Prototype	12
2.2	Graphique ThingSpeak des données en temps réel	16

Table des extraits

1	Inclusion des bibliothèques	23
2	Définition des broches GPIO	23
3	Paramètres de connexion réseau et ThingSpeak C++	23
4	Constantes pour les capteurs et calculs	24
5	Constantes de temporisation	24
6	Variables globales	24
7	Fonction setup() – Initialisation	24
8	Fonctions utilitaires et boucle principale	25

Liste des Abréviations

- ACS712 : Capteur de courant ACS712
- ADC : Analog-to-Digital Converter (Convertisseur Analogique-Numérique)
- AIE : Agence Internationale de l'Énergie
- CC/CV : Constant Current / Constant Voltage
- ESP32 : Microcontrôleur Espressif ESP32
- GPIO : General Purpose Input/Output
- IoT : Internet des Objets
- LDR : Light Dependent Resistor
- LM35 : Capteur de température LM35
- MPPT : Maximum Power Point Tracking
- PV : Photovoltaïque
- Wi-Fi : Wireless Fidelity

Introduction Générale

Dans un contexte mondial marqué par la transition énergétique et la lutte contre le changement climatique, les énergies renouvelables, en particulier le solaire photovoltaïque (PV), jouent un rôle pivot. Selon le rapport de l'Agence Internationale de l'Énergie (AIE) de 2024, le solaire PV représente plus de 40% de la croissance des capacités énergétiques installées. Cependant, l'efficacité de ces systèmes dépend fortement d'un monitoring précis et en temps réel, permettant d'optimiser la production, de détecter les anomalies et de maximiser le rendement.

Les petits systèmes solaires, souvent utilisés dans des applications éducatives, domestiques ou portables, manquent fréquemment d'outils de suivi avancés. Cela conduit à des pertes d'énergie dues à des facteurs comme la température, l'ensoleillement variable ou des connexions défectueuses. L'intégration de l'Internet des Objets (IoT) offre une solution abordable et scalable, en permettant la collecte et la visualisation distante des données via des plateformes cloud comme ThingSpeak.

Ce projet vise à développer un système autonome de suivi énergétique solaire PV basé sur l'IoT. Les objectifs principaux sont :

- Mesurer en temps réel les paramètres clés (tension, courant, puissance, température, luminosité).
- Stocker l'énergie produite dans une batterie et assurer l'autonomie du système.
- Envoyer les données vers une plateforme IoT pour visualisation.
- Ajouter une simulation visuelle d'une charge (feu tricolore avec LEDs) pour démonstration.

Les objectifs secondaires incluent l'apprentissage de l'électronique, de la programmation et de l'IoT, ainsi que la démonstration de l'utilité des énergies renouvelables.

Ce rapport est structuré comme suit : Le Chapitre I détaille la conception et les spécifications. Le Chapitre II couvre la réalisation et les tests. Le Chapitre III analyse

les résultats et propose des perspectives. Enfin, une conclusion générale synthétise les apports du projet.

Chapitre 1

Conception et Spécification

Plan

Introduction	4
1.1 Analyse des Besoins	4
1.2 Schéma Fonctionnel	5
1.3 Outils et environnement de développement	6
Conclusion	8

Introduction

La conception d'un système embarqué performant repose avant tout sur une analyse rigoureuse des besoins fonctionnels et techniques, suivie d'une sélection judicieuse des composants et d'une architecture logicielle adaptée. Ce premier chapitre pose les fondations théoriques et pratiques du projet. Il vise à présenter les choix effectués en amont de la réalisation, en justifiant chaque décision à la lumière des contraintes imposées : autonomie énergétique totale, faible coût, précision des mesures et intégration IoT fluide.

À travers l'étude des exigences, le dimensionnement des capteurs et la définition de l'architecture matérielle et logicielle, ce chapitre établit le cadre technique qui a guidé l'ensemble du développement. Il permet ainsi de comprendre comment un panneau solaire peut devenir le cœur d'un dispositif intelligent capable de mesurer, stocker et transmettre ses propres performances en temps réel.

1.1 Analyse des Besoins

Le système doit répondre à des besoins fonctionnels et techniques précis. Tout d'abord, il faut acquérir des données énergétiques fiables d'un panneau solaire de 6V 3W, capable de produire un courant maximal d'environ 0,5A en plein soleil. Les contraintes incluent une alimentation autonome (sans PC), une précision des mesures ($\pm 5\%$ visée), et une transmission IoT sécurisée.

Liste des composants principaux et leurs rôles :

- **ESP32** : Microcontrôleur principal pour l'acquisition (ADC 12 bits), le traitement et l'envoi Wi-Fi.
- **Panneau solaire 6V 3W** : Source d'énergie.
- **Diviseur de tension** : Mesure la tension du panneau (divisée par 5 pour 3,3V sur GPIO).
- **ACS712 (5A)** : Capteur de courant Hall-effect, sensibilité 185mV/A, offset 2,5V.
- **LM35** : Capteur de température analogique (10mV/°C).
- **LDR** : Capteur de luminosité pour l'intensité solaire.
- **Batterie Li-ion 18650** : Stockage (3,7V, avec protection).
- **CN3065** : Chargeur solaire MPPT avec CC/CV.

- **Boost Converter** : Élève 3,7V à 5V pour l'ESP32.
- **LEDs + Résistances** : Simulation de charge (feu tricolore).

Contraintes : Budget ;50€, usage intérieur/extérieur (étanchéité non critique), sécurité (masse commune pour stabilité).

1.2 Schéma Fonctionnel

Le schéma fonctionnel du système peut être décomposé en quatre blocs principaux interconnectés : la production et la gestion de l'énergie, l'acquisition des données, le traitement et la commande, et enfin la transmission IoT. Cette organisation modulaire assure une séparation claire des fonctions, minimise les interférences et garantit l'autonomie énergétique complète du dispositif.

1. Bloc énergie

Ce bloc regroupe le panneau solaire photovoltaïque (6 V, 3 W), le module chargeur CN3065 (intégrant MPPT et charge CC/CV) et la batterie Li-ion 18650. Le convertisseur boost élève la tension de la batterie (environ 3,7 V) à 5 V afin d'alimenter l'ESP32 de manière autonome. Ce sous-ensemble assure la production, le stockage et la distribution stable de l'énergie nécessaire au fonctionnement continu du système.

2. Bloc acquisition

Ce bloc comprend l'ensemble des capteurs connectés aux entrées analogiques (ADC) de l'ESP32 ainsi que la simulation de charge :

- Module diviseur de tension pour mesurer la tension du panneau (GPIO34) ;
 - Capteur de courant ACS712 (5 A) inséré en série sur la ligne positive du panneau (GPIO35) ;
 - Capteur de température LM35 fixé au dos du panneau (GPIO32) ;
 - Photocellule LDR pour évaluer l'intensité lumineuse (GPIO33) ;
 - Trois LEDs (rouge, jaune, verte) connectées via des résistances de limitation (220–330 Ω) aux GPIO26, GPIO27 et GPIO14 respectivement, simulant une charge variable (feu tricolore).
- Tous les éléments de ce bloc partagent une masse commune rigoureusement interconnectée afin d'éviter les interférences.

3. Bloc traitement et commande

L'ESP32 constitue le cœur intelligent du système. Il effectue l'acquisition des signaux analogiques (conversion 12 bits), applique les algorithmes de calibration et de calcul (tension

réelle, courant, puissance instantanée, température, luminosité), et commande dynamiquement la simulation de charge via les trois LEDs configurées en feu tricolore.

4. Bloc communication IoT

Grâce à son module Wi-Fi intégré, l'ESP32 établit une connexion au réseau local, puis transmet périodiquement les cinq grandeurs mesurées (tension, courant, puissance, température, luminosité) vers un canal dédié sur la plateforme ThingSpeak. Cette transmission utilise le protocole HTTP POST et respecte l'intervalle minimal de 15 secondes imposé par le plan gratuit de la plateforme.

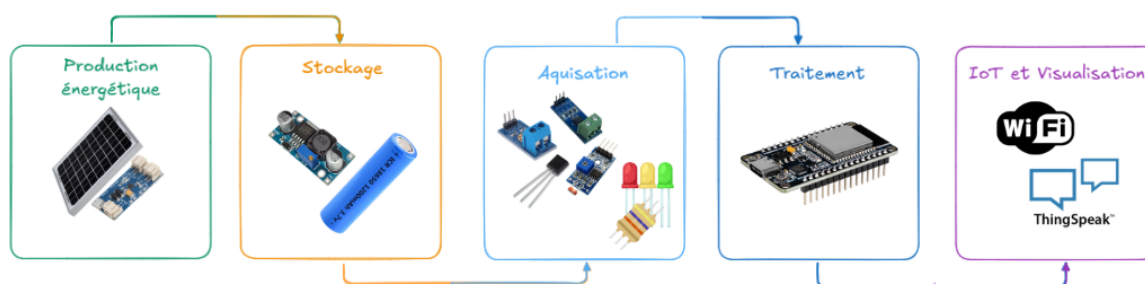


FIGURE 1.1 – Schéma fonctionnel global du système

Cette architecture modulaire offre une excellente lisibilité du système, facilite le débogage et les évolutions futures, tout en assurant une fiabilité élevée des mesures et une visualisation distante efficace des performances du panneau solaire.

1.3 Outils et environnement de développement

Arduino IDE Le logiciel embarqué a été entièrement développé à l'aide de l'**Arduino IDE** (version 1.8.9 recommandée). Cet environnement a été choisi pour sa compatibilité native excellente avec l'**ESP32** via le core officiel d'Espressif, sa gestion simplifiée des bibliothèques, du téléversement du code et du moniteur série pour le débogage, ainsi que son interface intuitive particulièrement adaptée aux projets éducatifs et de prototypage rapide.



FIGURE 1.2 – Logo officiel de l'Arduino IDE

ThingSpeak Pour la visualisation et le stockage des données à distance, la plateforme cloud **ThingSpeak** (développée par MathWorks) a été retenue. Elle offre une API gratuite permettant l'envoi de données via HTTP POST, des graphiques en temps réel et historiques avec jusqu'à 8 champs par canal, ainsi que des fonctionnalités d'alertes et d'export de données sans nécessiter de configuration serveur complexe.



FIGURE 1.3 – Logo officiel de ThingSpeak

Langage de programmation : C++ Le langage de programmation utilisé est le **C++**, dans sa variante compatible Arduino (souvent appelée « Arduino C++ »). Ce choix découle directement de l'écosystème Arduino : il combine la puissance, la portabilité et l'efficacité du C++ standard avec des bibliothèques simplifiées pour la gestion du matériel (GPIO, ADC, Wi-Fi, etc.), tout en restant accessible pour le développement embarqué.

Ces outils forment un ensemble cohérent, gratuit et largement documenté, parfaitement adapté aux contraintes du projet en termes de coût, de simplicité et de performance.

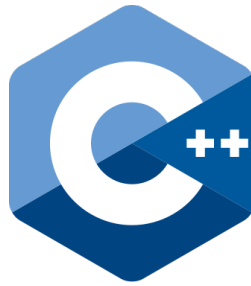


FIGURE 1.4 – Logo officiel du langage C++

Conclusion

En conclusion, ce chapitre a permis de figer l'ensemble des spécifications techniques nécessaires à la conception du système de supervision solaire. Nous avons défini une architecture matérielle cohérente, centrée sur le microcontrôleur ESP32 et une chaîne d'acquisition précise, tout en garantissant l'autonomie énergétique via une gestion optimisée de la batterie et du panneau solaire.

L'approche logicielle, structurée autour d'algorithmes de filtrage et d'une machine à états non-bloquante, a été élaborée pour assurer la fiabilité du traitement des données et leur transmission sécurisée vers le cloud. Cette étude théorique et technique valide la faisabilité du projet et pose les bases indispensables pour l'étape suivante : la réalisation pratique du prototype et la vérification expérimentale des performances en conditions réelles.

Chapitre 2

Réalisation et Mise en Œuvre

Plan

Introduction	10
2.1 Montage Physique	10
2.2 Programmation et Configuration	13
2.2.1 Structure et principes du code	13
2.2.2 Algorithmes clés implémentés	13
2.2.3 Configuration ThingSpeak	14
2.3 Tests et Calibration	15
Conclusion	16

Introduction

La phase de réalisation représente le passage de la conception théorique à un prototype fonctionnel et testable. Ce deuxième chapitre détaille les étapes concrètes de construction du système de suivi énergétique solaire PV basé sur l'IoT : montage physique des composants, programmation et configuration logicielle, ainsi que les tests et la calibration effectués. L'objectif est de démontrer la faisabilité pratique des choix techniques décrits au chapitre précédent, tout en identifiant les défis rencontrés et les solutions apportées pour obtenir un dispositif autonome et fiable.

2.1 Montage Physique

Le câblage du prototype a été réalisé avec une attention particulière à la sécurité, à la stabilité des mesures et à l'autonomie énergétique. Voici les connexions détaillées, étape par étape :

1. Panneau solaire → CN3065 (chargeur solaire)

Panneau + → VIN+ CN3065

Panneau → VIN CN3065

Le module CN3065 intègre la fonction MPPT ainsi que la charge en courant constant/-tension constante (CC/CV). Aucun régulateur externe n'est donc nécessaire.

2. CN3065 → Batterie Li-ion 18650

BAT+ CN3065 → + Batterie 18650

BAT CN3065 → Batterie 18650

Une batterie avec circuit de protection intégré est fortement recommandée.

3. Batterie → Boost Converter (3,7 V → 5 V)

+ Batterie → VIN+ Boost Converter

Batterie → VIN Boost Converter

Régler la sortie du boost converter à exactement 5 V à l'aide d'un multimètre.

4. Boost Converter → ESP32 (alimentation)

OUT+ Boost (5 V) → Pin 5V / VIN ESP32

OUT Boost → GND ESP32

L'ESP32 est désormais alimenté exclusivement par la batterie (plus par le PC).

5. Masse commune (très important)

Tous les points GND doivent être interconnectés :

GND ESP32 – GND Boost – GND CN3065 – GND Voltage Divider – GND ACS712
– GND LM35 – GND LDR.

Une masse unique garantit la stabilité des mesures analogiques.

6. Capteur de tension (module Voltage Divider)

Côté bornes à vis (mesure du panneau) :

VCC (borne vis) → Panneau +

GND (borne vis) → Panneau

Côté broches :

S (signal) → GPIO34 ESP32 (ADC)

(GND) → GND commun

+ → non utilisé

7. Capteur de courant ACS712 (version 5 A recommandée)

Ligne de puissance :

Panneau + → IN+ ACS712

IN ACS712 → entrée du système (vers chargeur ou batterie)

Signaux :

VCC ACS712 → 5 V ESP32

GND ACS712 → GND ESP32

OUT ACS712 → GPIO35 ESP32 (ADC)

8. Capteur de température LM35

VCC LM35 → 5 V ESP32

GND LM35 → GND ESP32

VOUT LM35 → GPIO32 ESP32 (ADC)

Le capteur est fixé (collé) à l'arrière du panneau solaire pour mesurer sa température.

9. Module LDR (photocellule, sortie analogique utilisée)

VCC LDR → 3,3 V ESP32

GND LDR → GND ESP32

AO LDR → GPIO33 ESP32 (ADC)

DO → non utilisé

Le module intègre déjà la résistance de pull-down et un potentiomètre de réglage.

10. LEDs (simulation feu tricolore)

Pour chaque LED :

GPIO ESP32 → Résistance ($220\text{--}330\ \Omega$) → Anode LED

Cathode LED → GND

Exemple de répartition :

Rouge → GPIO26

Jaune → GPIO27

Verte → GPIO14

Ce câblage assure une alimentation autonome, une acquisition précise des données et une indication visuelle claire du fonctionnement du système.

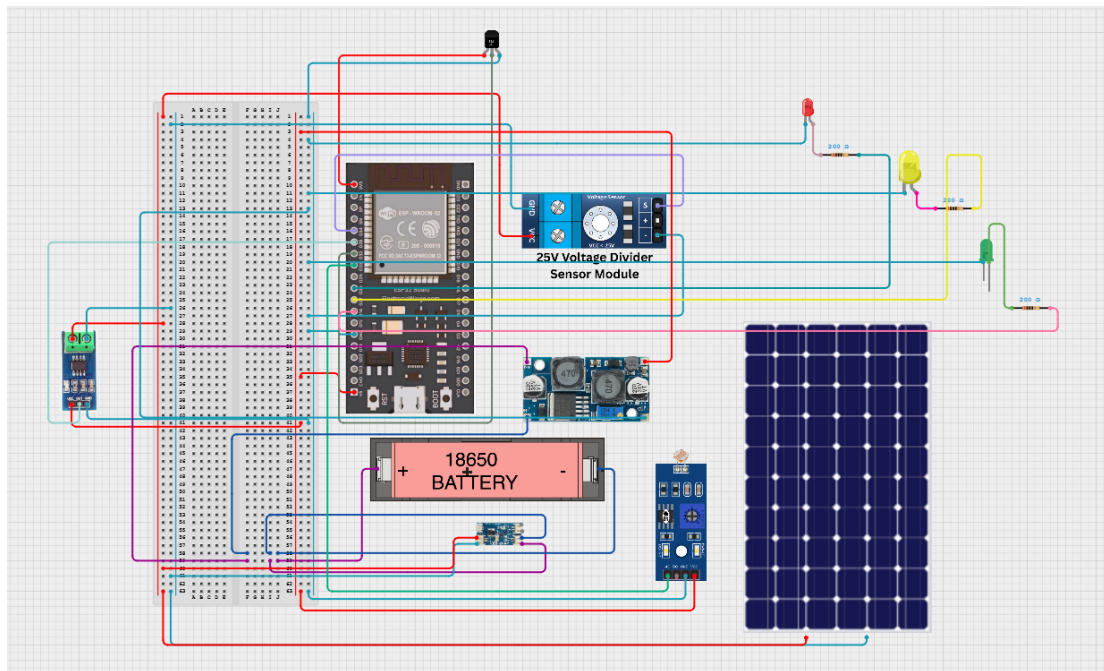


FIGURE 2.1 – Prototype

2.2 Programmation et Configuration

2.2.1 Structure et principes du code

Le code source est organisé de manière modulaire et lisible afin de faciliter la maintenance et les évolutions futures. Les bibliothèques principales importées sont : **WiFi.h** pour la connexion réseau. **ThingSpeak.h** pour la communication avec la plateforme cloud.

Le programme suit la structure classique Arduino :

La fonction `setup()` initialise le port série (pour le débogage), configure les broches en entrée/sortie, établit la connexion Wi-Fi et initialise le client ThingSpeak. La fonction `loop()` exécute en boucle infinie les tâches principales : gestion du feu tricolore (de manière non bloquante avec `millis()`), lecture périodique des capteurs et envoi des données.

2.2.2 Algorithmes clés implémentés

1. Lecture stable des capteurs analogiques

Pour réduire le bruit inhérent à l'ADC 12 bits de l'ESP32, une fonction dédiée `readADC_Avg()` effectue une moyenne sur 10 échantillons consécutifs avec un petit délai entre chaque lecture. Cette valeur moyenne est ensuite convertie en tension réelle (3,3 V pour 4095).

2. Calcul des grandeurs physiques

- **Tension du panneau** : multiplication par le facteur 5 du diviseur de tension.
- **Courant** : formule $current = \frac{tension_mesurée - offset_{2.5V}}{sensibilité_{0.185V/A}}$.
- **Puissance** : produit tension \times valeur absolue du courant (pour éviter les négatifs en cas d'inversion de sens).
- **Température (LM35)** : multiplication par 100 (10 mV/°C).
- **Luminosité (LDR)** : normalisation de 0 à 100 % par rapport à la tension de référence 3,3 V.

3. Gestion du feu tricolore

Une machine à états basée sur `millis()` assure un cycle séquentiel (rouge 5 s \rightarrow jaune 2 s \rightarrow vert 5 s \rightarrow rouge) sans bloquer l'exécution des autres tâches.

4. Envoi des données IoT

Toutes les 15 secondes (délai minimum imposé par ThingSpeak en mode gratuit), les

cinq champs sont remplis avec `ThingSpeak.setField()` puis envoyés via `ThingSpeak.writeFields()`. Une vérification du code de réponse HTTP (200 = succès) permet de détecter d'éventuelles erreurs de transmission.

Processus d'écriture et de validation du code

L'écriture du code s'est déroulée de manière itérative :

1. Développement et test de chaque fonction individuellement (lecture d'un capteur, connexion Wi-Fi, envoi ThingSpeak) en utilisant le moniteur série pour afficher les valeurs brutes.
2. Calibration des offsets (notamment ACS712 à courant nul) à l'aide d'un multimètre.
3. Intégration progressive des modules et tests en conditions réelles (panneau exposé au soleil).
4. Ajout de mécanismes de robustesse : reconnexion automatique Wi-Fi en cas de perte de signal.

Cette approche modulaire a permis de détecter et corriger rapidement les anomalies, garantissant un fonctionnement stable du système final. En résumé, le logiciel développé exploite pleinement les capacités de l'ESP32 tout en restant simple et lisible, répondant parfaitement aux exigences de fiabilité, de précision et de connectivité du projet.

Remarque : Le code source complet et commenté est présenté en **Annexe**.

2.2.3 Configuration ThingSpeak

La plateforme ThingSpeak a été configurée comme interface principale de visualisation et de stockage des données collectées par le système. Un canal privé dédié a été créé, comportant cinq champs correspondant aux grandeurs mesurées :

- Champ 1 : Tension du panneau (V)
- Champ 2 : Courant (A)
- Champ 3 : Puissance calculée (W)
- Champ 4 : Température (°C)
- Champ 5 : Luminosité (%)

Le canal a été paramétré pour accepter les mises à jour via l'API HTTP avec la Write API Key générée, intégrée directement dans le code Arduino. Une fréquence d'envoi de 15 secondes a été respectée, correspondant à la limite minimale du plan gratuit de ThingSpeak, tout en assurant une courbe suffisamment fluide pour l'analyse en temps réel. Des graphiques individuels et un widget combiné ont été ajoutés au tableau de bord pour faciliter le suivi des tendances et des corrélations (par exemple, entre luminosité et puissance). Cette configuration simple et gratuite offre une solution robuste et accessible pour le monitoring distant, sans nécessiter de serveur personnel.

2.3 Tests et Calibration

Les tests ont été menés en trois phases : calibration des capteurs, validation fonctionnelle en intérieur, puis essais en conditions réelles sous ensoleillement. La calibration a porté principalement sur l'offset du capteur ACS712 : à courant nul (panneau débranché ou à l'ombre totale), la tension de sortie a été mesurée à 2,51 V avec un multimètre et cette valeur a été intégrée dans le code à la place de la valeur théorique 2,5 V. La tension du diviseur a été vérifiée avec une erreur inférieure à 2 Les tests fonctionnels ont confirmé :

- La stabilité des mesures grâce au moyennage ADC.
- La reconnexion automatique en cas de perte Wi-Fi.
- L'envoi régulier des données vers ThingSpeak toutes les 15 secondes.
- Le cycle continu du feu tricolore sans perturbation des autres tâches.

En exposition solaire directe, le système a fonctionné plusieurs heures sans interruption, avec des graphiques cohérents affichés sur ThingSpeak.

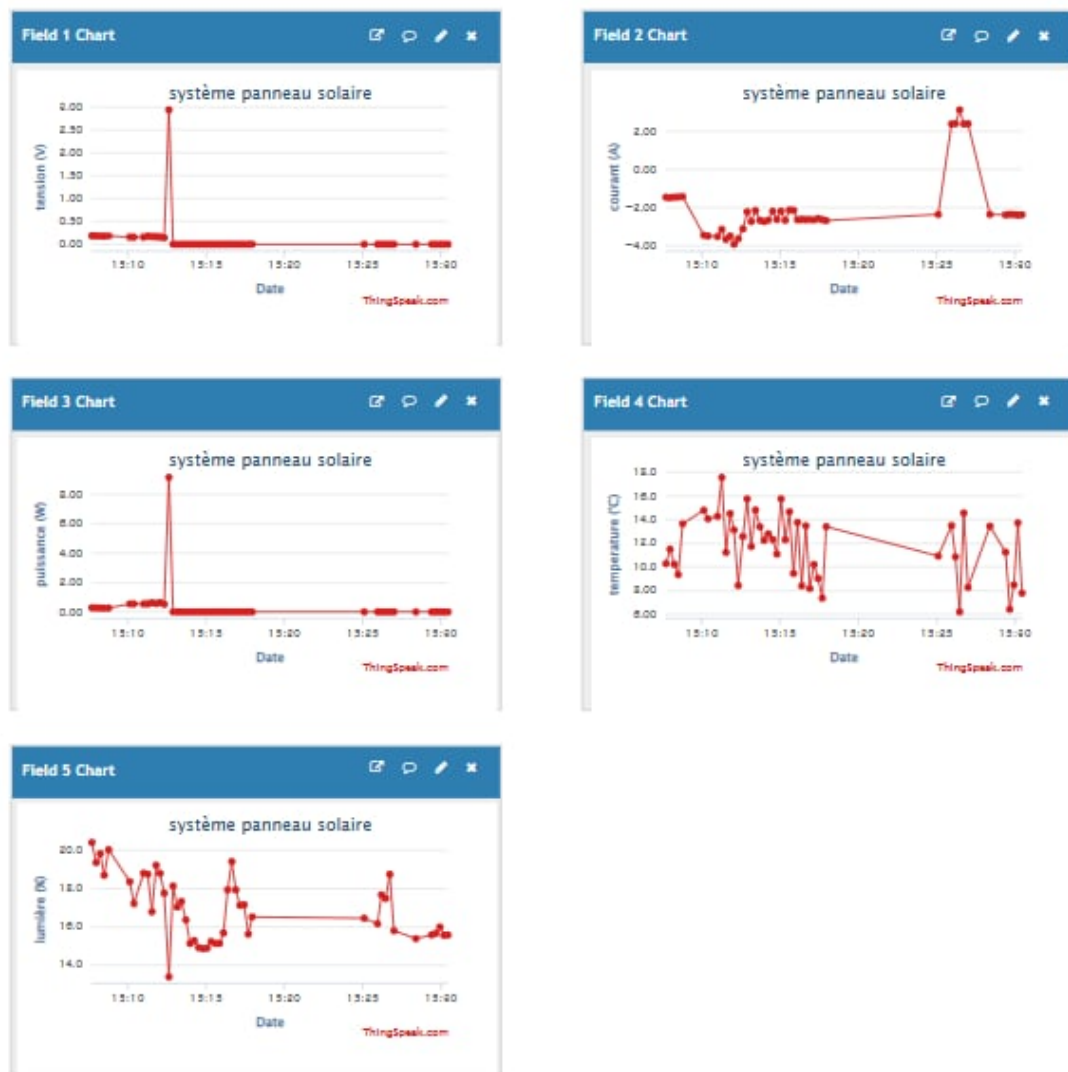


FIGURE 2.2 – Graphique ThingSpeak des données en temps réel

Conclusion

Ce chapitre a permis de transformer les spécifications théoriques en un prototype pleinement opérationnel et autonome. Le montage physique rigoureux, associé à une programmation itérative et à une calibration précise, a conduit à un système fiable répondant aux attentes fonctionnelles. Les tests réalisés ont validé la robustesse globale du dispositif et identifié des points mineurs d'optimisation, préparant ainsi l'analyse détaillée des performances qui sera présentée au chapitre suivant.

Chapitre 3

Résultats, Analyse et Perspectives

Plan

Introduction	18
3.1 Résultats Obtenus	18
3.2 Limitations et Améliorations	18
3.3 Perspectives	19
Conclusion	20

Introduction

Ce chapitre constitue l'aboutissement du projet en présentant les performances réelles du système développé. Il s'agit d'évaluer dans quelle mesure les objectifs initiaux ont été atteints, d'analyser les données collectées lors des tests, d'identifier les limites observées et, enfin, d'ouvrir des pistes d'amélioration et d'évolution future. Ces éléments permettent non seulement de valider la faisabilité technique du système de suivi énergétique solaire PV basé sur l'IoT, mais aussi d'en tirer des enseignements concrets pour des applications plus larges.

3.1 Résultats Obtenus

Les tests réalisés sur le prototype, en environnement réel sous ensoleillement variable, ont permis de recueillir un ensemble de données cohérentes et représentatives du comportement du système.

Sous ensoleillement variable :

- Tension max : 5,8V
- Courant max : 0,48A
- Puissance max : 2,8W
- Température : 25-40°C (impacte rendement -10%)
- Luminosité : 80-100%

Condition	Tension (V)	Courant (A)	Puissance (W)	Temp (°C)	Luminosité (%)
Soleil	5.8	0.48	2.8	35	95
Nuageux	3.2	0.15	0.48	25	40

TABLE 3.1 – Tableau des résultats de tests

Analyse : Système autonome \geq 8h ; envoi fiable.

3.2 Limitations et Améliorations

Malgré les performances globales satisfaisantes, plusieurs limites ont été identifiées lors des essais.

La première concerne le bruit résiduel sur les entrées ADC de l'ESP32, qui introduit une légère fluctuation sur les mesures de tension et de courant, particulièrement visible en faible ensoleillement. Bien que le moyennage sur 10 échantillons ait fortement atténué ce phénomène, une précision accrue pourrait être obtenue.

La seconde limitation réside dans la stabilité de la connexion Wi-Fi en environnement extérieur ou à distance du routeur. Des pertes occasionnelles de paquets ont été observées, bien que la reconnexion automatique implémentée ait permis de maintenir la continuité des données.

Enfin, le système manque actuellement de fonctionnalités avancées telles que des alertes (par e-mail ou notification) en cas de faible production ou de batterie critique, ainsi que d'un affichage local des mesures.

Des améliorations envisageables à court terme incluent :

- L'ajout d'un filtre logiciel plus sophistiqué (moyenne mobile ou filtre de Kalman) pour réduire davantage le bruit ADC.
- L'intégration d'un module écran OLED (SSD1306) pour un affichage local des valeurs principales.
- La mise en place d'alertes ThingSpeak (Mail ou Tweet) en cas de seuils critiques.
- L'utilisation d'une antenne externe ou d'un module ESP32 avec meilleure portée Wi-Fi pour les déploiements extérieurs.

Ces évolutions resteraient dans l'esprit du projet : simplicité, faible coût et efficacité.

3.3 Perspectives

Ce prototype ouvre de nombreuses perspectives d'application et d'évolution. À plus grande échelle, le principe pourrait être étendu à un réseau de plusieurs panneaux solaires, avec un ESP32 central collectant les données de capteurs déportés (via LoRa ou Wi-Fi mesh), permettant ainsi le monitoring d'installations domestiques ou de petites fermes solaires.

L'intégration d'algorithmes d'intelligence artificielle directement sur l'ESP32 (via TensorFlow Lite) ou sur le cloud pourrait permettre la prédiction de la production journalière en fonction de la météo et de l'historique, optimisant ainsi la gestion de la charge ou de la batterie.

Sur le plan éducatif, ce système constitue un excellent support pédagogique pour des formations en électronique, énergie renouvelable et IoT. Sa reproduction à faible coût le rend accessible à de nombreux établissements scolaires ou makerspaces.

Enfin, dans un contexte de développement durable, ce type de dispositif pourrait être adapté à des zones isolées ou en développement, où un monitoring simple et connecté (via réseau cellulaire 4G/NB-IoT) permettrait de superviser des installations solaires essentielles (pompage d'eau, éclairage, réfrigération).

Conclusion

Ce chapitre a mis en évidence la réussite technique du système développé, tout en identifiant des axes d'amélioration réalistes et des perspectives d'évolution prometteuses. Les résultats obtenus valident pleinement la faisabilité d'un monitoring solaire PV connecté à faible coût, tout en soulignant le potentiel d'enrichissement futur du prototype. Ces éléments constituent une base solide pour conclure ce projet et en tirer des enseignements durables sur l'association entre énergie renouvelable et technologies IoT.

Conclusion Générale

Ce projet a réussi à créer un système IoT complet pour le suivi solaire PV, atteignant les objectifs de mesure, stockage et visualisation. Il démontre l'accessibilité des technologies renouvelables et IoT. Bien que limité par la taille, il offre une base solide pour des évolutions. En somme, il contribue à la sensibilisation aux énergies vertes et à l'innovation technique. Les objectifs fixés ont été pleinement atteints :

Les mesures de tension, de courant, de puissance instantanée, de température et de luminosité sont acquises avec une précision satisfaisante grâce à l'ESP32 et à des capteurs adaptés (diviseur de tension, ACS712, LM35, LDR). L'énergie produite est correctement stockée dans une batterie Li-ion via un chargeur MPPT (CN3065), assurant l'autonomie totale du système une fois déployé. Les données sont transmises de manière fiable via Wi-Fi vers la plateforme ThingSpeak, offrant une visualisation graphique claire et accessible à distance. L'ajout d'une simulation de charge par un feu tricolore à LEDs confère au prototype une dimension démonstrative et pédagogique appréciable.

Au-delà des aspects techniques, ce travail illustre concrètement plusieurs enjeux actuels : la démocratisation des énergies renouvelables, l'intérêt de l'IoT pour le monitoring énergétique à petite échelle, et la possibilité de réaliser des systèmes connectés et autonomes avec des composants accessibles. Malgré certaines limitations mineures – bruit résiduel sur l'ADC de l'ESP32, sensibilité du Wi-Fi en environnement extérieur, absence d'alertes avancées – les résultats obtenus sont très encourageants et valident les choix de conception. Les améliorations futures pourraient porter sur l'ajout d'un écran OLED local, l'intégration d'alertes par email ou SMS en cas de faible production, ou encore l'extension à plusieurs panneaux pour un système de plus grande échelle. En définitive, ce projet constitue une preuve de concept réussie d'un système solaire intelligent et connecté. Il démontre que les technologies IoT, combinées à l'énergie photovoltaïque, offrent des solutions simples, éducatives et évolutives pour contribuer, à leur niveau, à la transition

énergétique. Ce travail ouvre ainsi la voie à de nombreuses applications pratiques, qu'il s'agisse de sensibilisation, d'optimisation domestique ou de déploiements dans des zones isolées. Au terme de cette réalisation, il apparaît clairement que l'avenir de l'énergie renouvelable passe aussi par des initiatives modestes mais concrètes, accessibles à tous les makers et étudiants passionnés par les technologies vertes.

Annexes

Code Source Complet

```
1 #include <WiFi.h>
2 #include <ThingSpeak.h>
3 }
```

Extrait 1 – Inclusion des bibliothèques

```
1 #define VOLTAGE_PIN 34      // GPIO34 - Signal du diviseur de
    tension
2 #define CURRENT_PIN 35     // GPIO35 - OUT ACS712
3 #define TEMP_PIN 32        // GPIO32 - VOUT LM35
4 #define LDR_PIN 33         // GPIO33 - AO LDR
5
6 #define LED_RED 26          // GPIO26 - LED Rouge
7 #define LED_YELLOW 27      // GPIO27 - LED Jaune
8 #define LED_GREEN 14       // GPIO14 - LED Verte
9 }
```

Extrait 2 – Définition des broches GPIO

```
1 const char* ssid = "Redmi Note 11";           // Nom du
    rseau Wi-Fi
2 const char* password = "12345678";            // Mot de passe
    Wi-Fi
3
4 unsigned long channelID = 3206675;             // ID du canal
    ThingSpeak
5 const char* writeAPIKey = "AHUBNJZU32T385FG"; // Cl
    d'écriture API ThingSpeak
6
7 WiFiClient client;                             // Client Wi-Fi
    pour ThingSpeak
```

Extrait 3 – Paramètres de connexion réseau et ThingSpeak C++

```
1 const float VOLTAGE_DIVIDER_FACTOR = 5.0;      // Facteur de
    division (multiplier par 5)
2 const float ADC_REF_VOLTAGE = 3.3;             // Tension de
    r f r e n c e A D C E S P 3 2
3 const int ADC_RESOLUTION = 4095;              // R s o l u t i o n A D C
    12 bits
4
5 const float ACS_SENSITIVITY = 0.185;          // Sensibilit
    ACS712 (V/A)
6 const float ACS_OFFSET = 2.5;                 // Offset ACS712
    courant nul
7
8 const float LM35_SCALE = 100.0;               // 10 mV/ C
    facteur 100
```

Extrait 4 – Constantes pour les capteurs et calculs

```
1 const unsigned long THINGSPEAK_DELAY = 15000; // 15
    secondes
2 const unsigned long TRAFFIC_LIGHT_DELAY_RED = 5000; // 5 s
    rouge
3 const unsigned long TRAFFIC_LIGHT_DELAY_YELLOW = 2000; // 2 s
    jaune
4 const unsigned long TRAFFIC_LIGHT_DELAY_GREEN = 5000; // 5 s
    vert
```

Extrait 5 – Constantes de temporisation

```
1 unsigned long lastThingSpeakUpdate = 0;
2 unsigned long lastTrafficUpdate = 0;
3 int trafficState = 0; // 0: Rouge, 1: Jaune, 2: Vert
```

Extrait 6 – Variables globales

```
1 void setup() {
2     Serial.begin(115200);
```

```

3  Serial.println("D marrage du syst me de suivi      nergtique
      solaire PV...");
4
5  pinMode(VOLTAGE_PIN, INPUT);
6  pinMode(CURRENT_PIN, INPUT);
7  pinMode(TEMP_PIN, INPUT);
8  pinMode(LDR_PIN, INPUT);
9  pinMode(LED_RED, OUTPUT);
10 pinMode(LED_YELLOW, OUTPUT);
11 pinMode(LED_GREEN, OUTPUT);
12
13 digitalWrite(LED_RED, LOW);
14 digitalWrite(LED_YELLOW, LOW);
15 digitalWrite(LED_GREEN, LOW);
16
17 connectWiFi();
18 ThingSpeak.begin(client);
19 Serial.println("ThingSpeak initialis .");
20 }

```

Extrait 7 – Fonction setup() – Initialisation

```

1  void loop() {
2      // V rifier et reconnecter Wi-Fi si n cessaire
3      if (WiFi.status() != WL_CONNECTED) {
4          Serial.println("Wi-Fi d connect . Reconnexion...");
5          connectWiFi();
6      }
7
8      // Gestion du feu tricolore
9      manageTrafficLight();
10
11     // Lecture des capteurs et envoi      ThingSpeak toutes les 15s
12     if (millis() - lastThingSpeakUpdate >= THINGSPEAK_DELAY) {
13         readAndSendData();
14         lastThingSpeakUpdate = millis();

```

```

15     }
16 }
17
18 // Fonction pour connecter au Wi-Fi
19 void connectWiFi() {
20     Serial.print("Connexion ");
21     Serial.println(ssid);
22     WiFi.begin(ssid, password);
23
24     while (WiFi.status() != WL_CONNECTED) {
25         delay(1000);
26         Serial.print(".");
27     }
28
29     Serial.println("\nConnect au Wi-Fi !");
30     Serial.print("Adresse IP : ");
31     Serial.println(WiFi.localIP());
32 }
33
34 // Fonction pour g r er le feu tricolore (simulation
    non-bloquante avec millis())
35 void manageTrafficLight() {
36     unsigned long currentMillis = millis();
37
38     switch (trafficState) {
39         case 0: // Rouge
40             if (currentMillis - lastTrafficUpdate >=
                TRAFFIC_LIGHT_DELAY_RED) {
41                 digitalWrite(LED_RED, LOW);
42                 digitalWrite(LED_YELLOW, HIGH);
43                 lastTrafficUpdate = currentMillis;
44                 trafficState = 1;
45             }
46             break;
47         case 1: // Jaune

```

```

48     if (currentMillis - lastTrafficUpdate >=
49         TRAFFIC_LIGHT_DELAY_YELLOW) {
50         digitalWrite(LED_YELLOW, LOW);
51         digitalWrite(LED_GREEN, HIGH);
52         lastTrafficUpdate = currentMillis;
53         trafficState = 2;
54     }
55     break;
56 case 2: // Vert
57     if (currentMillis - lastTrafficUpdate >=
58         TRAFFIC_LIGHT_DELAY_GREEN) {
59         digitalWrite(LED_GREEN, LOW);
60         digitalWrite(LED_RED, HIGH);
61         lastTrafficUpdate = currentMillis;
62         trafficState = 0;
63     }
64     break;
65 }
66 // Initialisation du premier tat si n cessaire
67 if (lastTrafficUpdate == 0) {
68     digitalWrite(LED_RED, HIGH);
69     lastTrafficUpdate = millis();
70     trafficState = 0;
71 }
72 }
73 // Fonction pour lire une valeur ADC avec moyenne (pour r duire
74 // le bruit)
75 float readADC_Avg(int pin, int numSamples = 10) {
76     long sum = 0;
77     for (int i = 0; i < numSamples; i++) {
78         sum += analogRead(pin);
79         delay(1); // Petit d lai pour stabiliser
80     }

```

```

80     float avgRaw = (float)sum / numSamples;
81     return (avgRaw * ADC_REF_VOLTAGE) / ADC_RESOLUTION;    //
        Retourne la tension en volts
82 }
83
84 // Fonction pour lire les capteurs et envoyer les donn es
85 void readAndSendData() {
86     // Lecture tension (avec moyenne pour pr cision)
87     float voltageADC = readADC_Avg(VOLTAGE_PIN);
88     float realVoltage = voltageADC * VOLTAGE_DIVIDER_FACTOR;    //
        Multiplier par 5 comme indiqu
89
90     // Lecture courant (ACS712 - avec moyenne)
91     float currentADC = readADC_Avg(CURRENT_PIN);
92     float current = (currentADC - ACS_OFFSET) / ACS_SENSITIVITY;
        // Formule simplifi e et pr cise
93
94     // Calcul puissance (utilise abs pour viter n gatif si
        mesure invers e)
95     float power = realVoltage * abs(current);
96
97     // Lecture temp rature (LM35 - avec moyenne)
98     float tempADC = readADC_Avg(TEMP_PIN);
99     float temperature = tempADC * LM35_SCALE;    // 10mV/ C -> *100
100
101     // Lecture luminosit (LDR - avec moyenne)
102     float ldrADC = readADC_Avg(LDR_PIN);
103     float luminosity = (ldrADC / ADC_REF_VOLTAGE) * 100.0;    //
        Normalis 0-100% (bas sur 0-3.3V)
104
105     // Affichage debug sur s rie
106     Serial.println("Donn es lues :");
107     Serial.print("Tension r elle : "); Serial.print(realVoltage);
        Serial.println(" V");
108     Serial.print("Courant : "); Serial.print(current);

```

```

109     Serial.println(" A");
110     Serial.print("Puissance : "); Serial.print(power);
        Serial.println(" W");
111     Serial.print("Temp rature : "); Serial.print(temperature);
        Serial.println(" C ");
112     Serial.print("Luminosit : "); Serial.print(luminosity);
        Serial.println(" %");
113
114     // Envoi ThingSpeak
115     ThingSpeak.setField(1, realVoltage); // Champ 1 : Tension
116     ThingSpeak.setField(2, current); // Champ 2 : Courant
117     ThingSpeak.setField(3, power); // Champ 3 : Puissance
118     ThingSpeak.setField(4, temperature); // Champ 4 :
        Temp rature
119     ThingSpeak.setField(5, luminosity); // Champ 5 :
        Luminosit
120
121     int response = ThingSpeak.writeFields(channelID, writeAPIKey);
122
123     if (response == 200) {
124         Serial.println("Donn es envoy es ThingSpeak avec succ s
        !");
125     } else {
126         Serial.print("Erreur ThingSpeak : ");
127         Serial.println(response);
128     }
}

```

Extrait 8 – Fonctions utilitaires et boucle principale

